Unsupervised image classification
Mozgalo 2017

Domjan Barić, Matej Pavlović, Petar Perković

May 31st 2017

Contents

Contents

1 Introduction

Our task is unsupervised image classification. We got unlabeled set of 6889 images and had to group them by their content. The goal is to determine number of classes and put images in one of them without using any additional information about images in our solution.

We will show three methods which we used to extract feature vectors of images: Autoencoder, convolutional neural network and HOG descriptor. **Autoencoder** is a neural network which compresses image to vector of much smaller dimensions and tries to reconstruct the original image from that vector. We can interpret vector as meaningful characteristic of an image, meaning similar images have similar vectors. We tested a lot of different architectures of neural network and chose the most efficient one.

We used **Convolutional neural network (CNN)** for unsupervised learning. We created artificially labeled dataset by making various random transformations on image and taking various parts of image. On that artificially labeled dataset our CNN was trained for categorization and succeeded in finding characteristic vector. Again we tested various architectures and took the one which was the most efficient.

HOG descriptor is method used in computer vision and image processing, for object and edge detection. It looks at gradient of color in image in purpose of describing the image.

For clustering we used k-means. This simple, but fast algorithm makes ideal candidate for first clustering of some dataset. Some more advance algorithms may give better solution, but they are too selective and therefore unnecessary because we will filter our clusters later on. Optimal number of clusters is determined automatically by calculating score of clusters for different numbers of clusters and by choosing the one with optimal score.

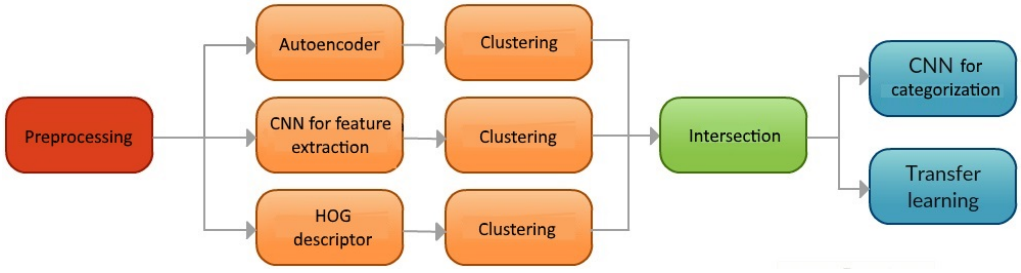

Figure 1: Solution pipeline.

To increase purity of clusters, we decided to make intersection between our methods. If the same image is in the same cluster after all three methods, we can say that is correctly clustered. As result we have very pure clusters, although the number of images is reduced.

In the end, we trained our CNN for classification by using new clusters. To comparison, we also tried fine tune Google's Inception V3 which resulted with CNN much smaller than Google's one, but results are comparable.

2 Methods

2.1 Image preprocessing

Images in the initial set are in different formats and different dimensions. All the pictures were converted to RGB and saved in JPG format. We noticed that a lot of images have monochrome frames. We removed them in order that our methods do not interpret them as a feature. Finally, we've changed the dimensions of all the images as desired.

2.2 Convolutional autoencoder

2.2.1 Autoencoder

Autoencoder is a type of neural network used to reduce data dimensions. Its essential character is symmetry. It consists of a part for encrypting and decrypting data. The encryption part extracts the most important characteristics of input. The middle layer can be understood as a data key. It represents essential input characteristics (key features). The decoding part attempts to reconstruct input from the middle layer as similar as possible .

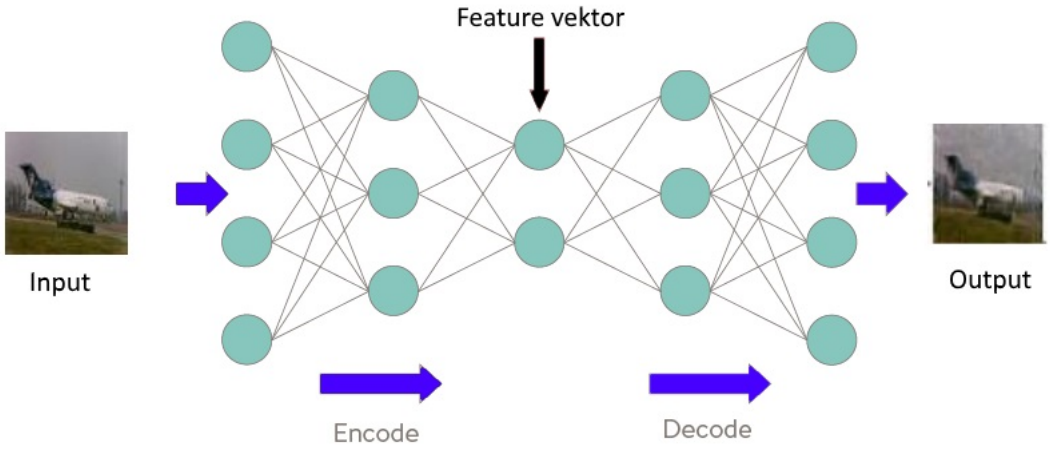

Figure 2: Autoencoder scheme

2.2.2 Convolutional autoenvoder

The convolutional neural network is inspired by the frontal cortex of living beings and is currently the state of the art network for image classification. It consists of a convolutional, *max-pooling* and *dropout* layers. The convolutional and max-pooling layers interchange with each other while the dropout layer is placed in certain locations.

The problem with autoencoder is that it ignores the $2D$ image structure and forces each feature to be global because the layers are completely connected. However, the object recognition trend is to find the local characteristics of an object that are repeated throughout the whole input. We decided to use a convolutional autoenkoder so that for each image we get a characteristic vector that has a dimension much smaller than the image dimension and then perform further analyzes over these vectors.

Each convolution layer contains a certain number of filters of a certain dimension, each filter extracts another feature and shape from the image. Smaller size filters better see the details in the picture. The output of the convolutional layer, which has the *dimension of the filter* $\times$ *number of the filter*, is input for the next layer of neural network. In our network, we used 3×3 filters because we wanted for our neural network to better match details and shapes in images, rather than the background.

The Max-pooling layer serves to reduce the dimensionality of the data. It works by taking the maximum of the nonoverlapping input subdivisions.

We've used 2×2 subdivisions. Max-pooling is sensitive to patterns and after size reduction, it tries to keep the pattern.

Reverse of max-pooling is the UpSampling layer that takes the input value and makes a matrix of that value, i.e. increases the dimension of the data. We increased the values to 2×2 matrices.

The dropout layer is used to prevent overfitting and works by randomly discharging some neurons in the network (label D). We used a dropout rate of 0.2.

After the convolutional layers, we added completely connected layers that further reduced the data dimension.

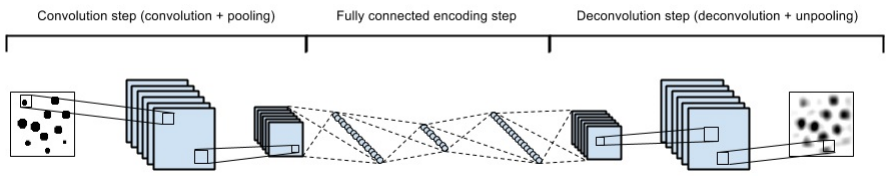

Figure 3: Scheme of convolutional autoencoder.

2.2.3 Network architecture

After testing multiple different network architectures we decided to:

M-32K-64K-128K-M-M-D - bf 150F - 512F-64K'-In-The-32K'-3K ',
Where K denotes convolution, K 'transposed convolutional, M is max pooling, D is a dropout, and F is a fully connected layer.

Output of *150F* layer is the characteristic vector of image. Inputs and outputs are in the form of 64 *times*64×3 because the input to our network is RGB image with dimensions 64×64. We came to the conclusion that this is the optimum image size because of its enough information and, on the other hand, not too big size for the neural network, which would make it more difficult to train. RGB image is represented by values between 0 and 255, the picture we normalized before sending it to the neural network at values between 0 and 1. For the activation functions we used everywhere *ReLu* except in the last layer where we used *sigmoid*, because we wanted to get a value between 0 and 1.

sigmoid function is defined as $f(x) = 1/(1 + exp(-x))$, and *ReLu* is defined as $f(x) = max(0, x)$ As a cost function, we used binary crossentropy.

The neural network contains 600 000 training parameters. The network is trained on the *Nvidia Geoforce 840M* graphics card. To run one epoch it took 40 seconds, and converged after 40 iterations.

2.3 Unsupervised feature learning with convolutional neural network

Deep convolutional networks have proven to be an excellent tool for learning image features. From the unlabeled set of images, we took section of 500 images and created artificially labeled image sets with various transformations of them. The convolutional net is trained on these labeled images. Later on, this convolutional network was used to obtain characteristics from the initial set of images.

2.3.1 Creating labeled image set

We've set the dimension of all initial images to 96×96 pixels. From these images we randomly picked 64×64 patches around the center of the picture. After that, transformations on the patches were made, and all the patches from the same image were placed in one class.

The transformations consist of:

- Translation: Vertical and Horizontal Translation up to 0.2 Sample Size
- Rotation: image rotation up to 20 °
- Scaling: multiplying the size of the patch with factor between 0.8 and 1.2
- Contrast: change the contrast for a factor between 0.5 and 2
- Color: Changing the color intensity for a factor between 0.5 and 2
- Color 2: Add value between -0.1 and 0.1 hue components in HSV color representation

If necessary, transformations can be added and removed. We took 500 patches and received 20 images from each transformation.

Figure 4: Several different transformations of the same patch.

2.3.2 Convolutional Neural Network

Using the obtained set of images, we trained the convolutional neural network. The network consists of 2 convolutional layers with 64 filters, followed by a fully connected layer of 128 units. Finally, *softmax* is a layer that has a number of units equal to the class number and serves as the network output.

The filter dimensions are 5×5 and after each convolution network there is a *Max pooling* layer of 2×2. Prior to the *softmax* layer, a dropout layer was added to prevent overfitting. Everywhere, except in the last layer, *ReLu* activation has been used. Deeper networks have learned more advanced features, but these features would separate already well-clustered groups to subclusters before they learn how to separate unclustered images. The network is constructed and trained within *Kerasa* on TensorFlow. Trained using *categorical crossentropy* as a cost function.

Once the convolutional neural network was trained, we extracted values in a fully connected layer for the entire original set of images. Later, we used these values as characteristic vectors to group images.

2.4 Histogram of oriented gradient (HOG)

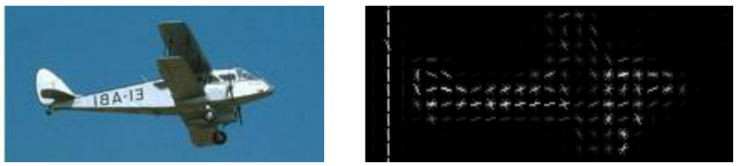

Figure 5: Image before and after HOG.

HOG finds essential image features, as it is used in computer graphics and image processing to detect objects. The gradient histogram finds the gradient intensity distribution, meaning it detects the edge in the image. It divides the image into less connected regions, for pixels within each region finds a gradient, then calculates a gradient change across multiple regions.

The descriptor connects the characteristics of all regions. As an input to HOG, a black and white image is used, because HOG can not work on multiple channels of the same image in parallel. Input images were dimensions of 64×64, and as output we got a vector size of 1764. HOG finds the edges well, so it is expected that objects of the same type have similar edge characteristic.

2.5 Clustering methods

2.5.1 Why k-means?

Why k-means? Although there are more advanced algorithms that can track cluster structure, they are generally slower (up to 100 times slower) and require many parameters to make the result a valid one. It would be necessary to change the parameters from one set of data to the other. The result is not universal. For example, *DBSCAN* has two parameters:

- *epsilon* - the radius of the ball about point where we look for other points
- *minPts* - The smallest number of points needed to be considered a cluster

These parameters need to be set separately for each dataset. Data needs to be monitored in order to determine the best parameters.

Additionally, we tried to use the *OPTICS* clustering algorithm, which is another unsupervised algorithm. It only needs one parameter and does not assume that the clusters are spherical. It follows the way from one point to another so you can find clusters of various shapes. The problem is that our data is densely distributed and there is always a path that contains all the images.

We want to separate the data without supervising the process. The inner structure of the cluster is not that interesting to us as we want to get pure separate groups. For this, *k-means* is ideal. It's fast, simple and ideal for first clustering. The assumption is that the points are grouped into spheres, and we expect that similar images will be centred around the same point. For *k-means*, only the number of clusters is required and we can specify it. As a result, we have simple and fast clustering without using any information about the nature of the data. We could use more advanced subgroup algorithms in the given clusters, but then we would control the data.

2.5.2 Finding the number of clusters

K-means algorithm minimizes the squares of distances of the cluster members from the centre of cluster, so we decided to use a rating that works on a similar principle. First, determine the average of the distance of cluster members from their center as follows:

$$intra = \frac{1}{N} \sum_{i=1}^{K} \sum_{x \in C_i} |x - z_i|^2,$$

Where z $_i$ is the center of ith cluster, K is the total number of clusters, and N is the total number of vectors we cluster.

What we also needed to keep in mind was the cluster distance between each other, so we did so by taking the distance of the two closest clusters. This means all other clusters will be further away.

$$inter = min(|x - z_i|^2) \text{ za } i \neq j;\ i, j \in [1, K]$$

Now we can define the clustering score as: $score = intra/inter$. The goal is to keep clusters as far apart and as compact as possible. Then clustering is better when *score* is lower.

In order to determine the number of clusters, we must determine the minimum and maximum number of clusters and calculate the score for everything between them. For artificially generated data, we take the global minimum as the optimal number of clusters. In the "natural" data, the cluster number corresponding to the global minimum after the first local maximum is taken.

If k is the number of clusters for which: $score(k-1) < score(k) > score(k+1)$, we say that k is the local maximum. Now let's look at which $k' > k$ gives the second local minimum of *score* and we take this k as the optimal number of clusters.

We opted for this method because it is possible to fully automate it. We can always determine that the minimum number of clusters is 1, and the maximum can be tenths of the dataset size. This is in accordance with unsupervised machine learning.

2.6 Intersection

The methods we use are very fast and can be trained in a shorter time on a regular laptop, but have a bit lower precision and would not be suitable

for retraining the last layer for transfer learning. However, each one of the methods better sees and describes the different image characteristics.

The autoencoder takes the whole picture and looks at the whole picture at the output. Therefore, it describes the background very well because it occupies most of the picture. The convolutional network takes various parts of the image, not the whole picture, so it better notices the details.

The idea is to use the best features of all methods. If there is a cluster containing the same images in all three methods, we can consider it to be well clustered.

We ended up with a vector in which the element number indicates the name of the image, and the value of the element in the field indicates which category it belongs to. Thus, vector has the form: $[0, 1, 3, ..., 7, 0, 6]$, which means that the first and second image fall into the same category. For each cluster in the first method we search for which cluster in the other two it matches. Using the other two methods, we clean the clusters from the first.

We noticed that the best-performing methods are the autoencoder and convolutional network, while a slightly less accurate result is given by *hog* classifier.

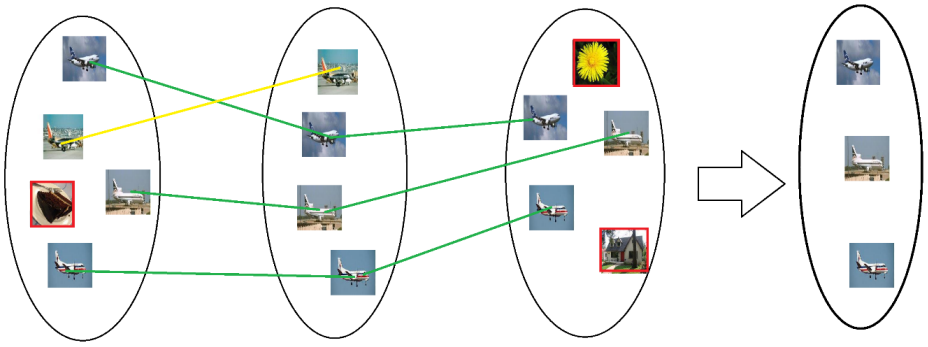

Figure 6: We try to find the picture from the first cluster in the other 2 clusters. If we find it, these images go into the final "pure" cluster (marked with a green line in the picture). If the picture is found in two clusters, but not in the third one, that image is not considered well clustered.

The problem occurs when the cluster in one method is dispersed in other methods. This may happen that the cluster in the autoencoder is separated into two equally sized clusters in the convolutional network. Therefore, as a

relevant number, we took the number of correctly clustered images divided by the size of a cluster.

If the cluster **0** in the auto-encoder contains 700 images and there are two relevant clusters in the convolution network, say cluster **2** and cluster **5** containing 1000 and 350 images. Let's have the number of images in intersection **0-2** 400 images, and in intersection **0-5** 300 images. Then the ratios of the number of images in intersection and cluster size for these two sets are 0.4 and 0.86. Therefore, the cross-section **0-5** is taken as the correct cross-section.

The procedure is repeated for all clusters. Finally, we have a smaller set of pictures, but we can guarantee the purity of each cluster. Very small clusters are automatically rejected.

2.7 Convolutional neural network for classification

After the intersection of clusters obtained by the initial methods we get a set of images that has very pure categories and is suitable for training the neural network.

We used a convolutional neural network with 2 convolutional layers with 64 filters of dimension 3×3, followed by a fully connected layer of 128 units. The last layer is the softmax layer with the number of units equals the remaining number of clusters after the intersection. After the convolutional layers there are maxpooling layers of 2×2 and after a fully connected layer there is a dropout layer with a rate of 0.5.

As the image set is reduced after the intersection, we use ImageDataGenerator from Keras to enlarge the image set. We train the network using categorical crossentropy as a cost function. The resulting network was saved and used to sort the initial set of images.

2.8 Transfer learning

Transfer learning is a method that uses already pre-trained networks, such as Google *InceptionV3*, which is trained with images on imageNET. These networks are very large and their training requires large amounts of resources and time. Their architecture is very complex (Inception has 48 layers, while our autoencoder has 9 layers). The advantage is that it allows them great

precision. *InceptionV3* has 2048 classes. For example, it differentiates the zebra from dalmatian or jaguar from cheetah.

We have no computing power to train something like that, but we can change the last layer and train it for our arbitrary classes. This works because such networks extract very important image features (feature extraction). Although they are trained with images that can be significantly different from the pictures we want to cluster, they are so powerful that they can learn about new, significantly different categories. This can be something specific, ranging from recognizing different types of meadow flowers to distinguishing galaxies from the stars.

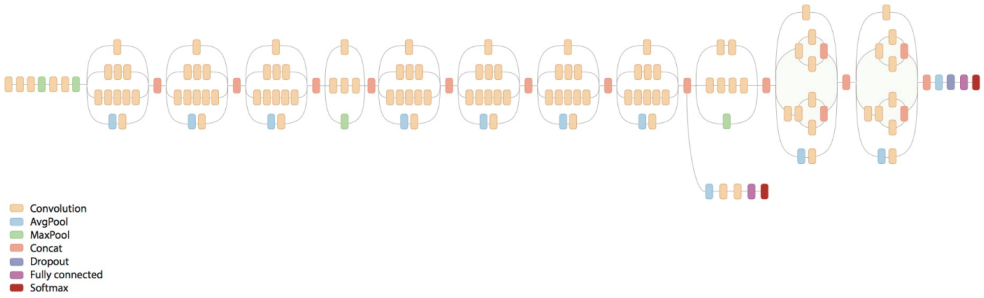

Figure 7: InceptionV3 architecture.

Our task is unsupervised clustering of images, at its core, and manual labeling of images is not allowed. We have to find a way to separate pictures in clusters without looking at them.

2.8.1 Fine tuning or retraninga a final few layers

If we want to retrain *InceptionV3* or any other network, the accuracy of the data is important. If we were to train the network on clusters that we got by one of our methods, the result would not be great due to the large cluster impurities. If we want to train network about the car and the images consist of 70% car, 20 % of buildings and 10 % leopards, we cannot expect the desired accuracy. However, using the intersection method, our clusters are pure, although the number of images is reduced. This does not represent a problem because we can increase the number of images by using modifications on the images, while maintaining the purity of the clusters. Therefore, with a high purity of clusters we can align the grid.

We used *InceptionV3* because it gives excellent results on the *Benchmark* image sets. In addition, it is well documented, its architecture is known, and is easily accessible. All the relevant data and required codes can be found in a short time.

We've retrained *InceptionV3* using *Keras*. First step is download of *InceptionV3*. We added *GlobalMaxPooling2D* layer to it. Then we added a completely connected layer. As an activation function we use the *ReLu* function(*rectified linear unit*). Furthermore, we added another dense layer with *softMax* activation that sorts the image into our desired categories.

Since we cannot and do not even need to re-train the whole network, all layers except added ones are frozen: we just train the added layers. It is convenient to use transfer learning when the set of images is not large, as in this case after the intersection. In order to increase the number of images we are training on, the images are mirrored, zoomed in and deformed. After adding the layers are trained, we train the convolutional layers that preceded the added layers.

Figure 8: Fine tuning final few layers.

The model is being saved after every epoch. As training can be very different from problem to problem, we advise you to do this manually. The map containing training images must be structured so that the images from the same section are within the same subfolder.

3 Results

3.1 Image preprocessing

All images are converted to RGB and saved to JPG format. White borders are cropped out, so that our methods do not interpret it as characteristic. Depending on method, images are being resized to different values.

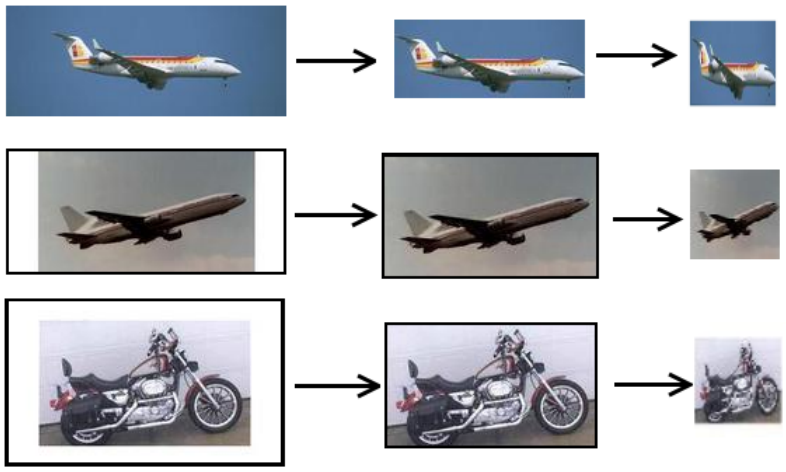

Figure 9: Preproccesing of images.

3.2 Convolutional autoencoder

It is obvious that reconstructed images are more blurry than the original ones. That is because of *max-pooling* layers in NN. They take only maximum value of 4 neighbour pixels, so we loose information about sharpness of the image. The sharpness of the image is not that important to us as objects can be recognized by its shape.

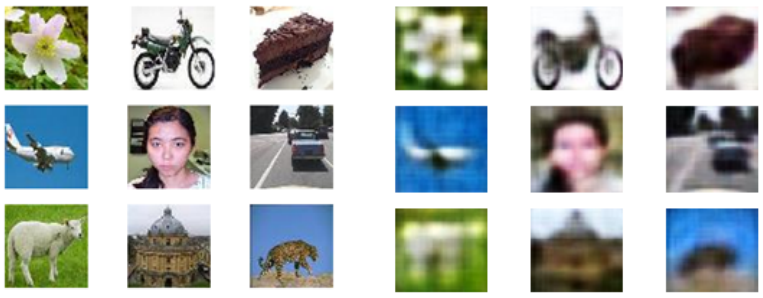

Figure 10: Example of images passing through the autoencoder. Left - original images, right - reconstructed images

While passing through middle layer, image is represented by only 150 values (from $64 \times 64 \times 3 = 12288$ at the beginning) which tells us that these are the most important image characteristic and it is possible to reconstruct original image. We chose architecture which gave the best ratio of accuracy of reconstruction and number of values in the middle layer.

The visualization of first convolutional layer can help observe how NN sees different shapes. Warmer colors on filter mean that these are characteristic which filter observes. Also we can see that there are some filters responsible for background, while others are responsible for object and its structure. Filters where sheep is blue are responsible for background and vice versa. Filters in deeper parts of neural network would represent simpler shapes from which the whole image is built.

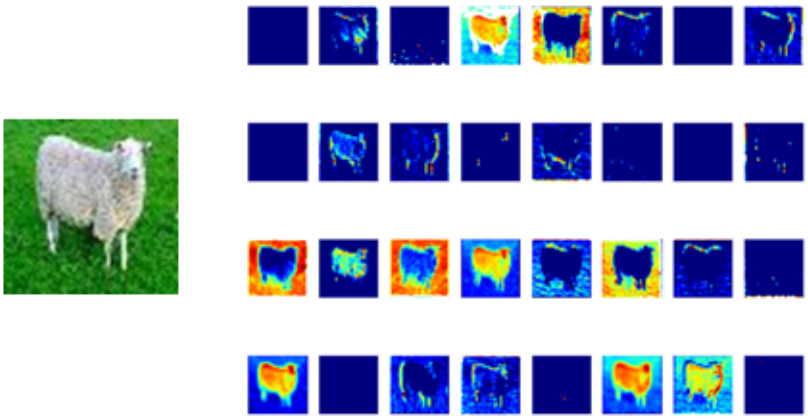

Figure 11: Left - input image, right - visualization of first convolution layer

After normalization and clustering, images with bad silhouette score are removed from clusters. The optimum number of clusters is 10. This method clusters well cakes, buildings, faces, flowers, planes and cars. Sheep and leopards are poorly recognized because animals are found in various poses. As for unsupervised classification, it is very difficult to learn that this is the same thing. Furthermore, there is a cluster with all the dark pictures, because of the black color on them that is much more dominant than shapes in the picture.

3.3 Unsupervised feature learning with convolutional neural network

3.3.1 Making artificially labeled set of images

The picture shows how several classes of the resulting set of images look like. The images obtained help the neural network learn to recognize the basic shapes of the image.

Figure 12: 4 images of 4 classes of the resulting labeled image set.

3.3.2 Convolutional neural network

Convolutional neural network whose architecture is described is trained.

After clustering, it can be noticed that the method correctly classifies motorbikes, buildings, cakes, planes, cars, human faces and flowers. They are recognized well because of the similarity of shapes in all images of the same category. Its biggest problem is sheep and leopards due to their diversity of poses and viewing angles. The accuracy of this method is 40%.

3.4 Histogram of the oriented gradient (HOG)

We got 9 clusters for this method. This method clusters well planes, cars, faces, buildings, motorbikes, and flowers, while animals are being clustered less accurate. Also, buildings and cakes are placed in the same category for a similar shape, as this method is based only on the shape of the object.

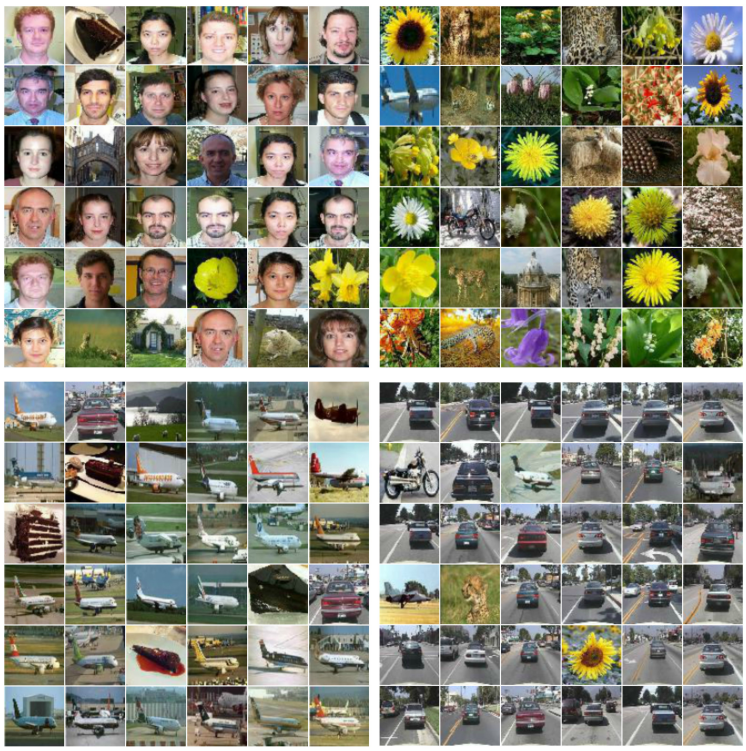

Figure 13: A few images from the face cluster (up-left), flowers (top-right), planes (bottom left) and cars (bottom-right) obtained by HOG method.

3.5 Intersection

After we have clustered the images with different methods, we want to observe the intersection of clusters to get idea how good are they for transfer learning and CNN for categorization.

We have noticed that we have the best results if we use the first method for the autoencoder. As a result of the function we have maps containing clean 9 clusters and one containing the images with dominant green color (sheep and cheetahs). Exceptionally small clusters are ejected. They are the result of mixed clusters that are scattered through two other methods. Also, due to their size are not suitable for learning.

Each of the methods works better for some images. For example, the autoencoder finds very well faces while scraping airplanes in various categories. The Hog classifier separates a cluster containing only planes. We want to use the best features of each method. Intersecting provides us with clean

clusters suitable for NN.

We can notice that clusters of cakes, faces, cars, planes and motorbikes are all pure. There is not a single wrong image in them. The clusters of flowers and buildings have a purity of more than 95%. Although all images of the dataset are not contained in the folders that we have provided, we know that they are grouped correctly.

Cluster size is not a problem (unless it is minimal) as we can increase their number through various image modifications. Much more important is that the clusters we get are clean, and that's what we accomplished. The cluster's high clarity allows us to safely train the net.

The last cluster predominantly contains sheep and cheetah, which is a result of the dominance of green color and that animals are often in different shapes and forms. They are not "hard" like a car, which makes learning difficult. Subsequently, there are 2047 images in the clusters after intersection.

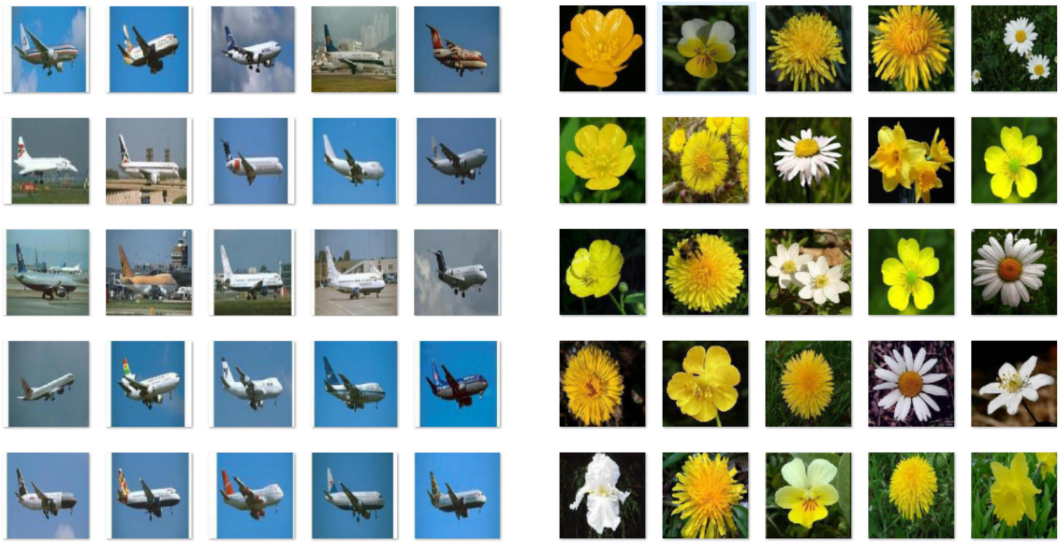

Figure 14: 2 clusters after the intersection. We have achieved a great cluster purity after the intersection.

3.6 Convolutional neural network for classification

We built a convolutional neural network that sorts the images in classes which we got after intersection. Network is a very good classification of faces, cars,

motorbikes, buildings, flowers, cakes and planes. In these categories, the network has a accuracy of at least 80%. Sheep and leopards are almost all in the same cluster as they were after the intersection. This neural network has been completely unsupervised and its results are much better than the initial methods. The architecture of this network is simple and the network itself is trained quickly. The accuracy of this method is 55%.

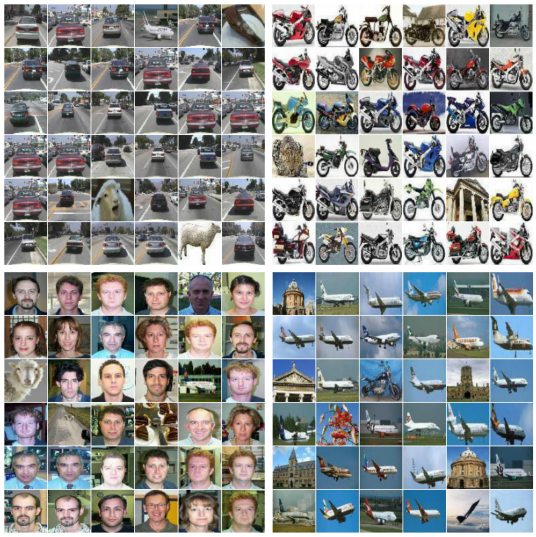

Figure 15: Some clusters after classification

3.7 Transfer learning

Without any fine tuning of Google's InceptionV3, with the use of k-means, it is possible to achieve 95% accuracy (this is because the network is most likely to be trained in these images, but this was not the point of our task). InceptionV3 is a huge network that requires more of computer resources to run than our network.

What makes it difficult is distinguishing leopards and sheep. Network will more likely separate two types of planes and two types of flowers, rather than separate cheetahs and sheep. If we cluster images as if there were 10 clusters, cheetahs and sheep would be mixed. Moreover, if we put it in 12 clusters, we get two clusters of planes and two clusters of flowers. At that moment, we should manually connect these clusters, meaning control the learning. We would stop with the unsupervised and begin with supervised learning which is not our task.

3.7.1 Fine tuning or retraninga a final few layers

After we have made the intersection, we have classes for fine tuning of the neural network. We have achieved the desired purity and we can change and train the last layer of InceptionV3.

After we trained our model (on the graphics card, for this dataset it takes up to an hour), we want to use it to classify the image.

As a result we have clusters of planes, faces, cakes, cars, motorbikes, flowers and buildings with high purity (over 95%). After intersection, the largest cluster had 370 images of cars. After training the network on these clusters, the smallest cluster has 434 images. The table shows the number of images after the intersection and after training the network.

We see that after training, the number of clustered images increased by more than 3.5 times on average. The cluster of cakes has increased the most, from 91 images to 665 images. The car cluster has not increased much, which means that our methods, which are faster and smaller than the InceptionV3, are good at clustering cars. The largest cluster matches images that are predominantly green. His growth can be explained because our model. If it was not sure in which cluster image should be putted, it would probably placed it in that (green) cluster.

We increased the number of images in the cluster and kept it clean. Using data after intersection and than fine tuning the network gave us the desired results. The process was completely unsupervised. At any point we did not use information about what is in the pictures, we just interpret the results. We have avoided long training and the need for great computing resources. The whole process can be done on a laptop with a graphics card within one work day.

4 Conclusion

"If intelligence was a cake, unsupervised learning would be the cake, supervised learning would be the icing on the cake, and reinforcement learning would be the cherry on the cake. We know how to make the icing and the cherry, but we don't know how to make the cake." - **Yann LeCun**, father of convolutional neural networks.

Why is unsupervised learning much harder than supervised? The reason is that a computer must only learn the data structure without any external information, and hence, have creative abilities. Although today there are excellent algorithms for unsupported learning, it is always easier and better, because of the available amount of data, to solve the problem with supervised learning.

Classification of images is a current problem in machine learning. Textit State of the art neural networks have an accuracy of more than 90% in the image classification. Such a result is currently unreachable for purely unsupervised learning. On given dataset, *InceptionV3* gives excellent results. This is a consequence of the complexity and size of the data on which the network is trained. Our problem is the unsupervised image classification. From the image, we must find essential features and then cluster them by these features.

The first method is to find the characteristic vector using Autoencoder. A neural network that works on the principle of encryption and decryption. The image enters the network, decreases to a much smaller vector and reconstructs the image from the vector. We want the reconstructed image to be as similar to the initial image. Similar images will have similar vectors, i.e. "passwords". After dozens of different architectures we have constructed and tested ourselves, we decided to give the best ratio of accuracy and speed. We managed to achieve accuracy of 38%. Although the autoencoder is neural network that requires learning and the data thus tagged, this is completely unsupervised learning. The exact result is the same as the input data. At any moment we did not enter any additional information or in any way monitored the data.

The second method we have made is the convolutional neural network. Convolutional networks are most often associated with supervised, rather than with unsupervised learning. What did we do? We took smaller parts of pictures that we deformed in different ways. From these smaller parts of the image we tried to determine the general characteristics of the images. Again we did not put in any additional information in the algorithm. The whole process is based on an unlabeled set of images that we have. We have also tried out a large number of different architectures that we built ourselves and decided for the best one. We got the accuracy of 40 %.

The third method we have observed is the HOG (histogram of oriented gradients). This is a method used in computer vision and image processing to detect the object in the image. In other words, the method looks at how

the colors change in smaller portions of the image. Again, this is completely unsupervised learning. The accuracy is 33%. It should be noted that we have observed many methods, but here we present three unsupervised methods that gave the best results.

After we extract the characteristic features from the image, we need to group them together. We have opted for the *k-means* method. Although there are more complex algorithms for finding clusters, they are usually significantly slower. The advantage of *k-means* is speed and simplicity, as it is most commonly used for initial clustering to reduce the problem to smaller parts. We have devoted ourselves more to finding and implementing the method for finding the optimal number of clusters.

We were not satisfied with the results we got, so we were thinking of increasing the accuracy. We realized that if the same image is in the same cluster in three different methods, we can safely say that it is well clustered. So we decided to find inter-methods. We identified which cluster from the first method corresponds to which clusters in the remaining two. We just took images that are in all three clusters. Effectively, we use the other two methods to clean the first. Let's note that this process is completely unsupervised because we only know to which cluster image belong and that information comes from *k-means*. As a result, we have very clean clusters, which almost do not contain any erroneous images. The only problem is that the number of images decreased to 2000. To increase the number of images, we built a convolutional neural network. We trained it on the data that "survived" intersection, and its task was to place the input images in one of these clusters. After training the network, we have input the entire dataset to cluster it. The number of images increased from two thousand to the total number of images in the set (6889). We kept the clarity of the cluster, but the number of images in the cluster increased more than three times. Even though we have trained a network, at any moment we did not look at the images and manually mark them and enter additional information. All the information that the network has received is a product of unsupervised methods. The accuracy of this method is 55 %.

Let's note again, until now we have used only unsupervised learning methods. The last thing we tried is transfer learning. We changed the last layers of *InceptionV3* and retrained them. The data on which we trained them are the same as in the convolutional neural network. The accuracy is 67 %. *InceptionV3* is a huge network that has been trained for a very long time, finds the characteristic features very well and is an excellent classifier. However,

the convolutional network we have built is much smaller and faster, and the results are equally good.

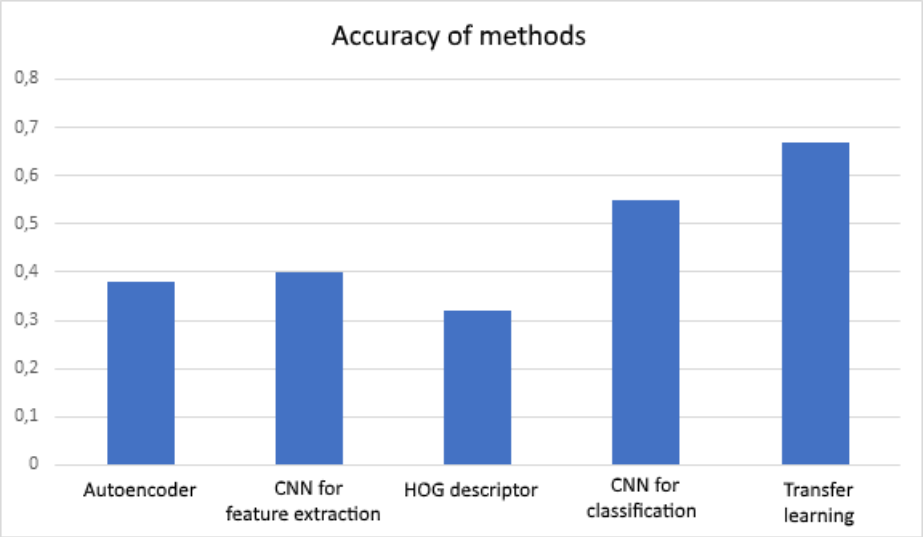

Figure 16: Comparison of the accuracy of the methods.

In the end, we managed from an unlabeled set of data, using three different, completely unsupervised methods: the *k-means* clustering algorithm and their intersection, to prepare an excellent dataset for training the neural network. At any moment we did not provide additional information to the problem. The whole process is completely unsupervised. Just as for the autoencoder and the first convolution network we used to extract the characteristic features of the image, we tried out many different architectures for the classification neural network, and as a product we got a network that gives an excellent result.