HOMEWORK 3 - MACHINE LEARNING COURSE
Borzillo Caterina, matricola 1808187

Problem: Dimensionality Reduction

One of the topics that really interested me about Unsupervised Learning technique is Dimensionality Reduction. As the name itself suggests, this type of problem deals with high-dimensional data, in particular with the number of input variables in a dataset. Usually, having a large number of features for input data is not bad since the fact that a lot of features give to the model a lot of information about the data, reason why the model should learn and predict more. But it is not always true; first, because if the number of features is high it may increase the variance of data e then the model could suffer from overfitting. Also, with a lot of variables, the characteristics of the input of data become very similar and so less meaningful from the point of view of classification. In the end, the point is that too many features are not useful, and that's why dimensionality reduction technique is used.

One issue about this method could be the information loss about the input data; this issue could be relevant, but sometimes looking at a dataset we can observe that there are some configurations of images that will never show up and for this reason are useless. For example, if we observe an airplane image, it will always have a particular shape which is recurrent in more or less all the dataset photos of the airplane class.

More in general, what dimensionality reduction wants to obtain is the transformation of input of many dimensions into a representation of input with much less dimensions, which is still very representative for the model. In fact, what the method wants to emphasize is the actual variability of each image data, in order to capture only the most interesting information about data itself. Intuitively, the aim of the method is to lose as least as possible information about the original input data and to do it there are various useful techniques.

The advantages of dimensionality reduction technique are the decrease of the training time and the reduction of the storage space required. These two factors are very useful and important when we deal with big dataset with tens of thousands images, or when we don't have the possibility to use GPU.

Dimensionality reduction is used in unsupervised learning problems that are problems in which the data in the dataset are not labelled. This means that the model couldn't train itself on images it already knows which class they belong to, and so must be able to learn and characterize a pattern to follow in order to classify well inputs.

To perform dimensionality reduction there are several methods we can use. In this homework I'll work on PCA and autoencoders.

Dimensionality reduction method: PCA

The most common and popular approach to dimensionality reduction is called Principal Component Analysis or PCA. As the definition says, PCA is based on the orthogonal projection of the data into a lower dimensional linear space, known as the principal subspace, such that the variance of the projected data is maximized. During the implementation we can choose the number of dimensions or principal components through which we want to reduce the input. Since PCA looks for a combination of features that capture well the variance of the original variance, the noise of the input images with a great probability will be reduced. After the reduction, the lower dimensional images will feed a model that will be trained and tested.

Dimensionality reduction method: Autoencoders

Another method used for dimensionality reduction problems is the autoencoder. Autoencoders are a branch of neural networks which has the aim of compressing the information of input data in a decreased dimensional space, called latent space, and then of reconstructing it as output. The key component is the "bottleneck", namely the lowest dimensional layer inside the deep network. The name autoencoder comes from the sum of two processes: the encoding step and the decoding step. The first is that one I'll use to train the model for the classification problem. In particular, once we "encoded" the input space in a lower dimensional space, I'll use it as input for my model in order to classify image data.

First step: importing libraries and loading data

As the first step, I imported some libraries and tools useful for problem resolution such as tensorflow, keras, matplotlib and sklearn. Then I checked if I was using a GPU.

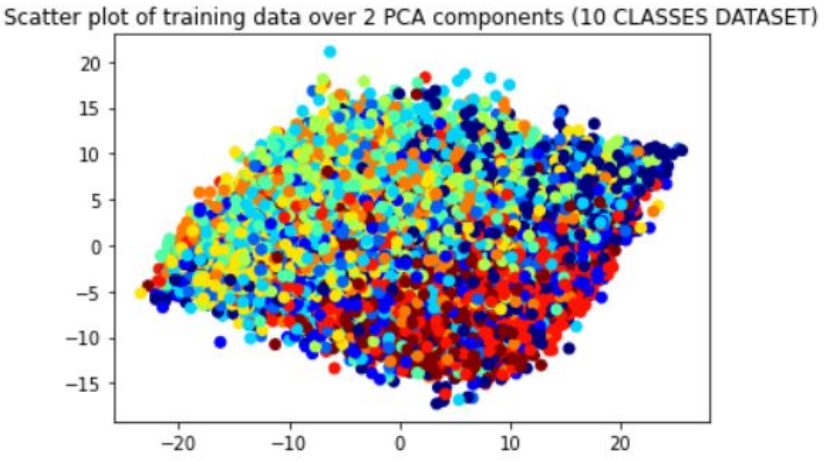
Dataset used

The dataset I used is the cifar10 dataset loaded through keras, which consists of 60.000 32x32 colour images divided in 10 classes with 6.000 images per class. Of these 6.000, 5.000 are training images and 1.000 are test images. The classes names are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck.

First attempt: dataset 10 classes

PCA dimensionality reduction method (10 classes)

As first attempt, I work on all 10 classes of the dataset, so on 60.000 images. Once I flattened each data input in a vector I observed that the number of components of each image in the dataset is 32x32x3 = 3072. Now, my objective is to make a reduction of these input images in order to capture only the main features on each sample. To see if the variability feature of these images was well defined through the classes I make a PCA transformation from 3072 dimensions to 2 dimensions and then I plotted the training data inputs as represented in a 2D plane.



The result is pretty confusing, so it means that the 10 classes are not well separated if projected and represented in 2 dimensions. For this reason, I perceived by intuition that the 10 classes of cifar10 dataset, even if reduced to a higher number of dimensions would not have given a good result as concerning the classification problem, because the differentiation between (for example) trucks and automobile would not permitted to the model to make acceptable predictions, considering also the fact that, even if we're dealing with a dataset which has the labels on input data, we want to solve an unsupervised problem due to which we'll not use the labels of images for training our model. But, to confirm the intuition I proceeded to manipulate the data, calculating the optimal number of reduced dimensions for which I have the 95% of variance through the images of the dataset. The number of principal components found is 658 for each input data (recall that the original dimension was 3072).

Evaluating performance with PCA reduction (10 classes - 658 components)

To see the results of classification on test data I used the K-Nearest Neighbors algorithm. This algorithm uses the neighbor samples information to predict the target class and it is based on identifying the nearest closer samples to a query example and on using those "neighbors" to determine the class of the query. The mechanism for assessing similarity is the distance. When we create the model based on this classifier we also have to specify the number of nearest neighbors to include in the majority of the voting process. For example is k = 10, the new data is classified by calculating the classes of the 10 closest data and then choosing the class to which the majority of data belong to. One method of deciding the number of k is to take the square root (sqrt(n)) of the number n of training samples. So I determine k equal to 300. After that, I use the predict function to classify the data on the test set and the accuracy obtained is about 6%, so very very very low.

For this reason what I predicted earlier about the confusing scatter plot made sense. For this reason I try to reduce the number of classes of this dataset and compute again the PCA reduction with the KNN predictions. But, before that, I wanted to see if using an autoencoder for reducing the dimensionality instead of PCA method gave better results.

Simple Autoencoder dimensionality reduction method (10 classes)

To achieve the task I create a very simple autoencoder made only by 1 hidden layer (called the latent space) with 658 units. This autoencoder has as input the image data of 32x32x3 dimensions and as output the same input shape. Also, inside the layers (the input one, the hidden one) I implemented the Dropout factor which is useful to prevent a model from overfitting; it consists on dropping out units in a neural network. But the peculiarity of autoencoders is the reconstruction from the latent space of the input images. An autoencoder which reconstructs the input image in a better way with respect to another, it's a favorable autoencoder. On the compile function I used as a loss function the mean squared error function which is obtained by dividing the term sum of squares of the residual error by the number of data. I used as metrics the loss value and adam as optimizer (adam is an optimization algorithm for stochastic gradient descent used usually for training deep network models).

Evaluating performance with encoder reduction (10 classes - 658 components)

After plotting one original image from input space and seeing the reconstruction made

by the simple autoencoder I divided the autoencoder structure into the encoder part and the decoder part in order to use the encoder output to evaluate performance of prediction. The encoder output will be the input space reduced by the encoder process and the dimension of this output is 658 that is the number of units of the unique hidden layer. Using again the KNN model to classify new test data I had as a result of accuracy the value of 12%. Too low.

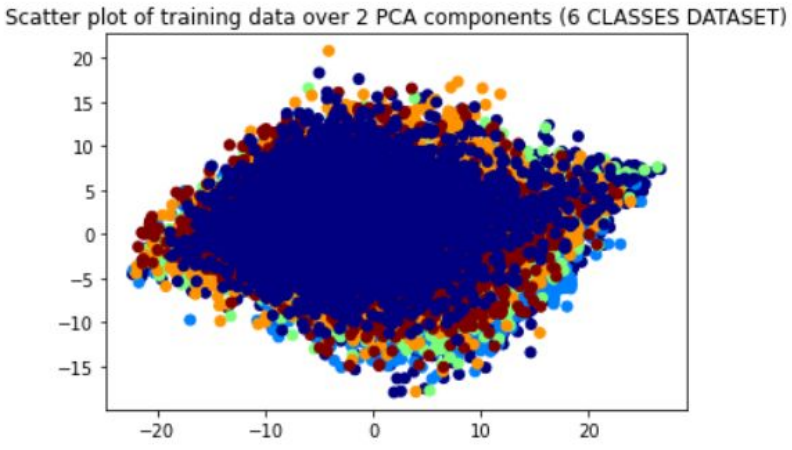
The reason why the accuracy calculated from the predictions of KNN model after have been reduced the input space with PCA and the simple autoencoder is that the classes of images of cifar10 were too similar and therefore in this case the reduction in the size of the input only created uncertainty and confusion.

Now, to improve the accuracy I tried to use not the entire dataset with the 10 types of classification but only 6 classes. The classes I pick up from the original dataset are the first 6 classes: airplane, automobile, bird, cat, deer, dog.

Second attempt: dataset with 6 classes

PCA dimensionality reduction method (6 classes)

Once I cut off the dataset from 10 to 6 classes, I immediately plotted the 2 dimensional graph to see if the data (this time, with the 6 classification problem) were separated. The scatter plot result is:



This is the result of the first 2 principal components of PCA reduction. It's possible to see that the data are more separated than before (10 classes dataset) but however the coloured points of different colors are still quite mixed.

What I expect is an increase of accuracy with respect to the previous study case with 10 classes but, a value accuracy that is, in general, still low.

So I calculated the ideal number of components for best PCA prediction for this dataset and I found the value of 631. It means that the number of components that better express the variance of input data for the dataset with 6 classes is 631. After calling the PCA function to represent each input image from 3072 to 631 features I create my K Neighbors Classifier model to evaluate performances by using reduction dimensionality technique.

Evaluating performance with PCA reduction (6 classes - 631 components)

Using as the number of neighbors in KNN model the number of the square root of the training sample (≈200) I fit the model and then predict the test data. The accuracy obtained is about 16% and we can see that it is better with respect to the previous case (6%) but it is still too low.

Then I proceeded as in the previous process, i.e. applying another dimensionality reduction technique which is the autoencoder. First I used the simple autoencoder with one hidden layer only.

Simple Autoencoder dimensionality reduction method (6 classes)

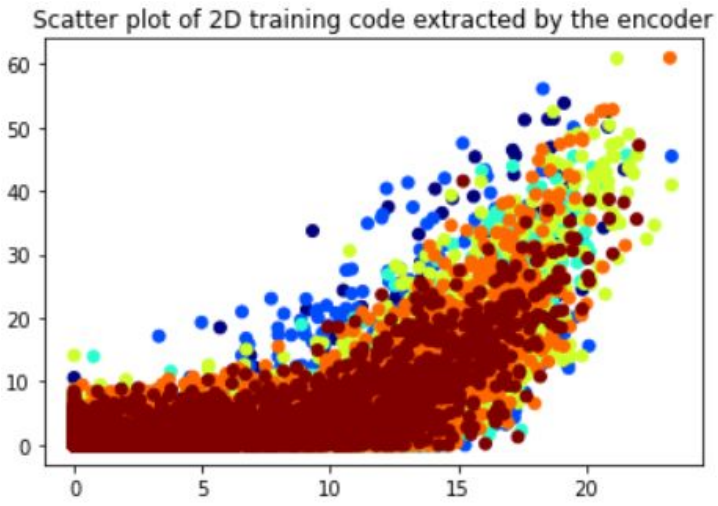
The number of the hidden layer units of this simple autoencoder is 631, which is the number of components I used for PCA evaluation. So in order to compare the results, I'll use the same reduced dimensionality. Using the same procedure as earlier (using the encoded reduction of inputs) , the accuracy obtained with KNN model with 631 reduced components is about 35% which is really better than the PCA evaluation with the same number of classes (16%).

Deep Autoencoder dimensionality reduction method (6 classes)

Since the autoencoder I used is very simple (with 1 hidden layer), I tried to create an autoencoder that had a more complex structure in terms of depth of the network, i.e. in terms of the number of hidden layers. So, this deep autoencoder has 8 layers: one of input, one of output and the rest are hidden layers. The "bottleneck" is the layer which has 300 units (according to the accuracy I found out that 300 units give the best accuracy

value). I implemented Dropout technique at every layer (to avoid overfitting). The activation function I used in every dense layer is the ReLU (Rectified Linear Unit) that is very commonly adopted in fully connected layers. At this point I compile the model with the loss function of mean squared error, with adam optimizer and then fit the model taking care about the loss metric.

I also made the scatter plot in 2 dimensions of the training data extracted by the encoder and the result is:



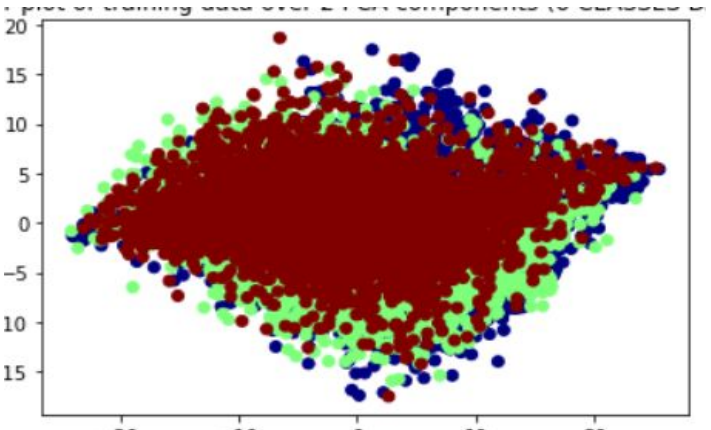
Here in the 2D graph, the separation of example is more evident.

The performance of the deep encoder reduction with KNN model sees an accuracy value of 43%. It is a result that is higher with respect to the simple encoder with one hidden layer. So this is a result that we expected, because this encoder has more fully connected layers and so in the latent space the dimensionality reduction is better done.

Third attempt: dataset with 3 classes

PCA dimensionality reduction method (3 classes)

As a last attempt I try to reduce even more the dataset. In fact, I extract from the original dataset 3 classes and I used them to compute the dimensionality reduction technique. I started plotting the 2 dimensional graph with the 2 principal components of PCA method. Now the situation is different from the first one, because the classes are only 3 and the colours associated to the classes are clearly distinguishable.



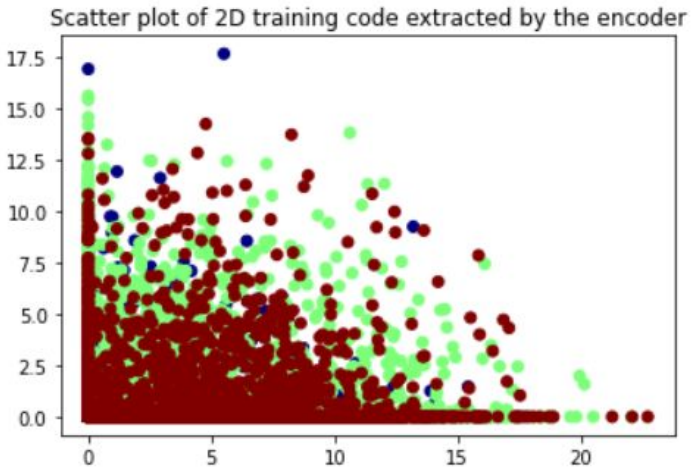
The first 3 classes are now: airplane, automobile and bird.

After finding the value that better expresses the variance of data for PCA dimensionality reduction (633), and after creating the KNN model as before with the number of neighbors equal to 130, I fit the model and then make a prediction with this model. The accuracy metric value is 54%. For a not optimal dataset for dimensionality reduction technique this is a good result, even if the model has to predict only 3 classes.

Simple Autoencoder dimensionality reduction method (3 classes)

About the simple encoder reduction with 3 classes the accuracy is up to 65%. We, in fact, already expected that this accuracy value would be better than the previous (PCA) accuracy result.

Deep Autoencoder dimensionality reduction method (3 classes)



In the figure there is a scatter plot of in 2 dimensions of the deep encoder reduction.

The points (3 colours) are distinguishable.

Finally the performance evaluation with deep encoder reduction method (with KNN classifier) computes an accuracy which is equal to 70%.

Final considerations

What I have learned studying and implementing this dimensionality reduction problem is that:

- PCA method reduces the input features in a way such that if I create a model with input the data reduced by PCA, the performance of predicting test data is low. In particular with a very large and various dataset the level of accuracy is unacceptable. But, in general, if we compare PCA reduction method and autoencoders reductions we see that the encoder process is more efficient and accurate.
- When I try to create a simple autoencoder (1 hidden layer) and then a deep autoencoder (with 8 hidden layers) we see the difference between the performances. That's clearly because a deep network reduces in a better way images with respect to a not deep network. However, the dataset with 10 classes, was unsuitable for dimensionality reduction problems because it was too large and too confusing. For example, with the MNIST dataset where the classes follow a very precise pattern of variability (the number one has always the same form) the results are better.
- Reducing the dataset from 10, to 6 and finally to 3 classes the results are very very different and the accuracy is so much better (obviously and intuitively) for models that have to classify images which belong to 3 classes.

Consideration on time complexity

From a computational point of view, what I experimented with this dimensionality reduction method is that the training time is significantly reduced compared to models that do not deal with the reduction of input data. The training time for 30 epochs of a model with 36.000 data with a batch size of 256 is 1 minute.

The reduced time complexity was in fact one of the best advantages of dimensionality reduction problems.