Unsupervised Clustering of Venus' Atmosphere UV Pictures

Alessandro Giovagnoli

Contents

1 Introduction

In this project machine learning techniques for clustering and classification are proposed to study satellite images of Venus atmosphere in order to attempt to provide a viable approach to extract useful information through remote sensing.

The project consisted of three main phases. In the first one, the data has been pre-processed. In particular, different space missions have been taken into account, namely *Venus Epress*, from ESA, and the *Akatsuki* satellite, from the Japanese space agency. In the end, the data from the Akatskuki probe has been used, in particular the UV camera pictures. They have been cropped, normalised, and an appropriate projection has been used to map the pixels from a sphere to the desired atlas.

In the second phase, machine learning techniques have been studied to find a proper solution to the problem of cloud classification with a ready dataset. First classical approaches like a supervised Convolutional Neural Network have been used. Then we moved on to unsupervised learning, trying to apply first classical techniques like KMeans, PCA, tSNE to learn how to autonomously divide the data in different classes, and then trying more advanced approches like autoencoders.

In the third phase, the results of the two previous parts have been put together, applying unsupervised techniques to the processed data of Venus atmosphere, with the goal of learning autonomously features of Venus atmosphere, clustering the different type of information available from each perspective.

2 Venus atmosphere

As a first task, Venus atmosphere has been studied in order to gather more information about its composition and structure, so to have a better understanding of the possible information that could have been extracted through the analysis of the images .

In the following we briefly report the most important characteristics.

2.1 Comparison between Earth and Venus

Comparison between Earth and Venus planetary characteristics.

	Earth	Venus
Equatorial Radius	6378.137 km	6050 km
Mass	5,972x10^24	4.867x10^24
Rotation period	23.93 h	243 days
Age	4.54 billion yr	4.5 billion yr

Comparison between Earth and Venus atmospheric characteristics.

	Earth	Venus
Atmospheric pressure at surface	1.01 bar	90 bar
Average temperature	14 °C	460 °C
Mass	$5.1480 \times 10^{1}8$kg	$4.8\times10^{2}0$kg
Density at surface	1.2 kg/m^3	67 kg/m^3

Comparison between Earth and Venus atmospheric composition characteristics.

	Earth	Venus
CO_2	0.038 %	96.5 % bar
N_2	78.084%	3.5%
O_2	20.948 %	0.001 %
H_2O	1 %	0.002 %

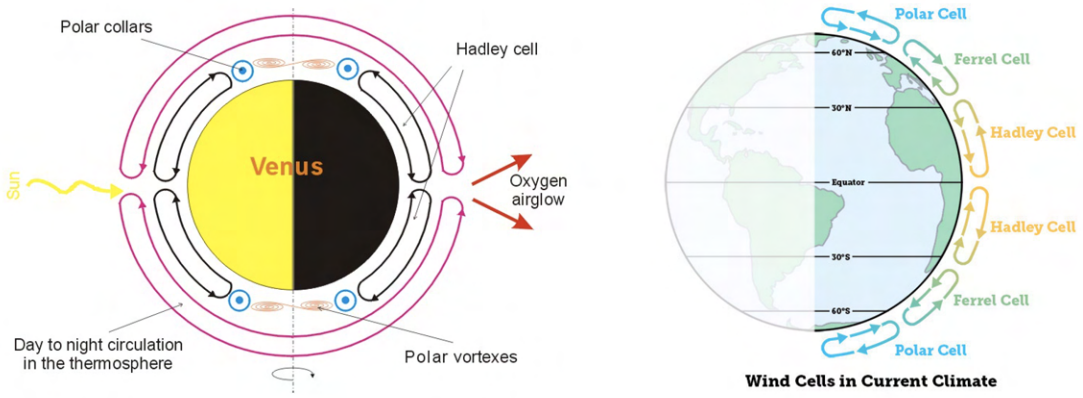

Figure 1: Conceptual comparison between the Earth's and Venus' atmospheres

It is also interesting to study the difference in Venus' atmospheric motion compared to the Earth one.

As shown in Figure 1, Venus atmosphere is in a state of vigorous circulation, where the different layers that it is composed of revolve around the surface. These layers have different circulation patterns, and, in particular, those closer to the surface revolve with a higher speed than those on the outside.

At the poles there are two big vortices, as shown in Figure 2, called polar vortices, wich are giant hurricane storms four times larger than their terrestrial analogues.

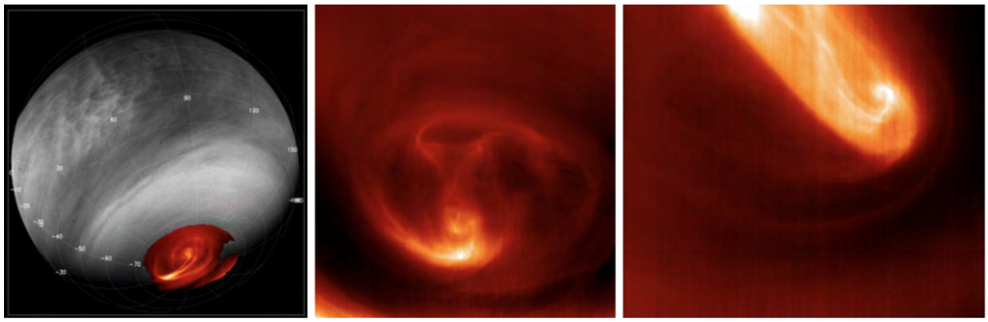

Figure 2: Pictures of the vortex at the poles of Venus' atmosphere

A particularly interesting phenomena is that the upper layer exhibits a phenomenon of super-rotation, in which the atmosphere circles the planet in just four Earth days, much faster than the planet's sideral day of 243 days.

This is of particular interest for the project because is means that the picture we see of the upper level of the atmosphere are not much related to the surface of the planet, since the rotation of the outer level is much faster than the sideral one.

2.2 Layer and characteristics

Venus atmosphere is made of three main layers: Mesosphere, Stratosphere and Troposphere. In Figure 3 a conceptual graph of the amosphere in shown.

The main layer, containing the clouds, the object of our study, is the Troposhpere. It extends up to 65 km At the top of the troposphere the temperature and pressure reaches Earth-like levels and clouds pick up speed to 100 m/s (360 km/h) The large amount of CO_2in the atmosphere together

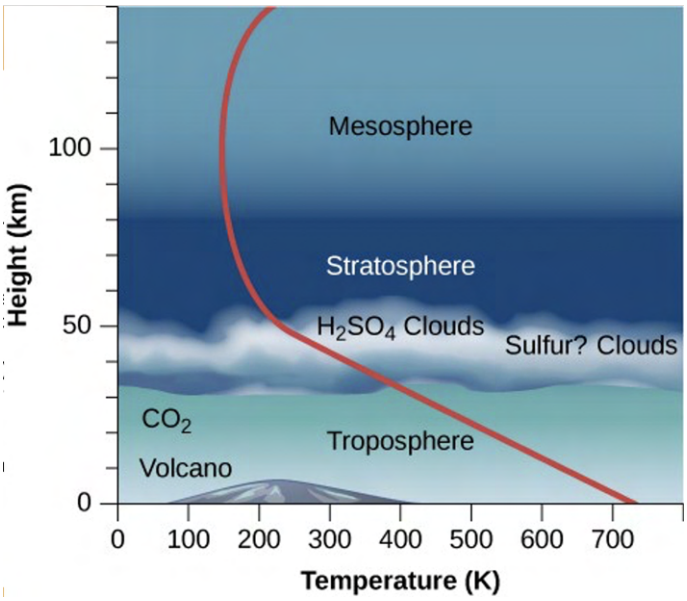

Figure 3: Graph of Venus' atmospheres layers and versus temperature

with water vapour and sulfur dioxide create a strong greenhouse effect, trapping solar energy and raising the surface temperature to around 740K (467°C) The troposphere on Venus contains 99% of the atmosphere by mass. At a height of 50 km the atmospheric pressure is approximately equal to that at the surface of Earth.

At the top of the Troposphere a thick layer of clouds is actually made of three parts, as shown in Figure 4

The highest goes from 58-68km, the middle one from 52-56km and the lowest from 48-52km. Thick and composed mainly (75–96%) of Sulfur Dioxide (SO_2) and drops of Sulfuric Acid (H_2SO_4). Although it rains, the rain never reaches the ground. The high temperatures evaporate the sulfuric acid drops, causing them to rise up again into the clouds, which have a yellowish colour. These clouds obscure the surface of Venus from optical imaging, and reflect about 75% of the sunlight that falls on them. The cloud cover is such that typical surface light levels are similar to a partly cloudy day on Earth. Their optical visibility is about 1 km.

Within the main cloud deck, particles of different sizes are found at different altitudes resulting in a layered substructure. Cloud particle sizes range from below 1 μm to more than 30 μm and basically show a trimodal size distribution. The smallest droplets, called "mode 1" particles form an aerosol haze extending throughout most of the cloud layer and their composition is not known yet. The intermediate "mode 2" droplets have typical diameters around 2-3 μm. The composition is of 75% H_2SO_4 and 25% H_2O. Most of the cloud mass is present in the biggest "mode 3" particles, which may be either evolved aggregates of sulphuric acid drops or another mode with a different composition.

2.3 Venus cloud pattern

We now study the main characteristics of the superficial cloud morphology. In Figure 5 a sketch of the visible, external atmosphere is reported.

This is a schematic representation of the basic clouds patterns on Venus when characterized by maximum tilt (left) and minimum tilt (right). The most important features are the Polar Cap, the Dark Polar Band, the Dark Mid-latitude Band and the middle Bow Shape.

We now provide several example of real images of UV photos of Venus atmosphere, visible in Figure 6 and 7.

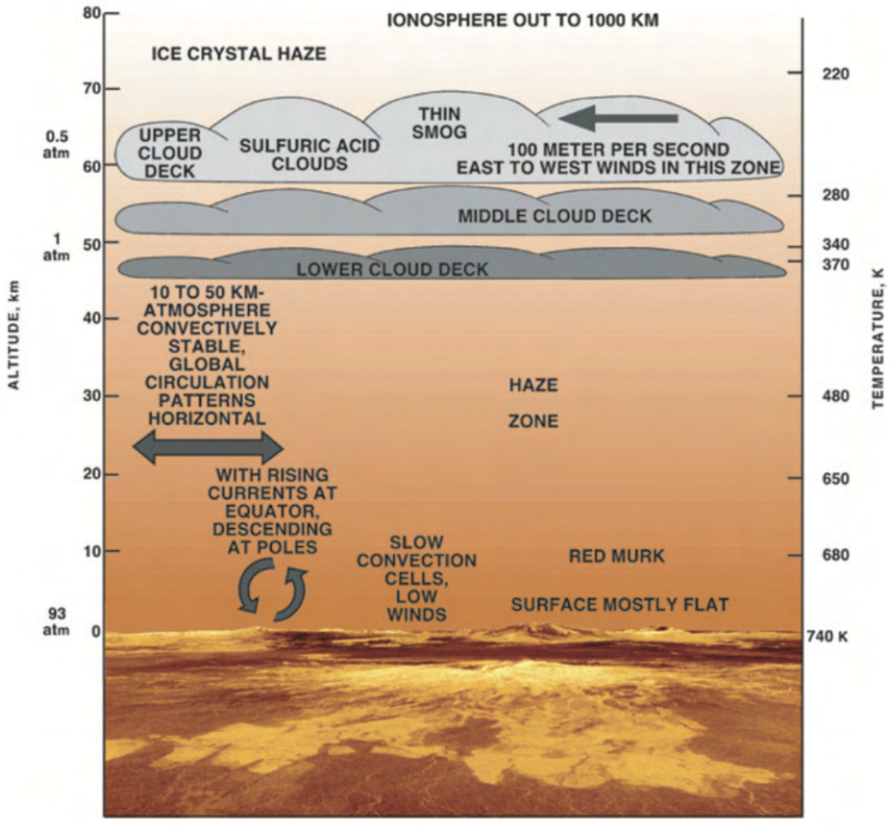

Figure 4: Graph of Venus' outside layer of the atmosphere more in detail

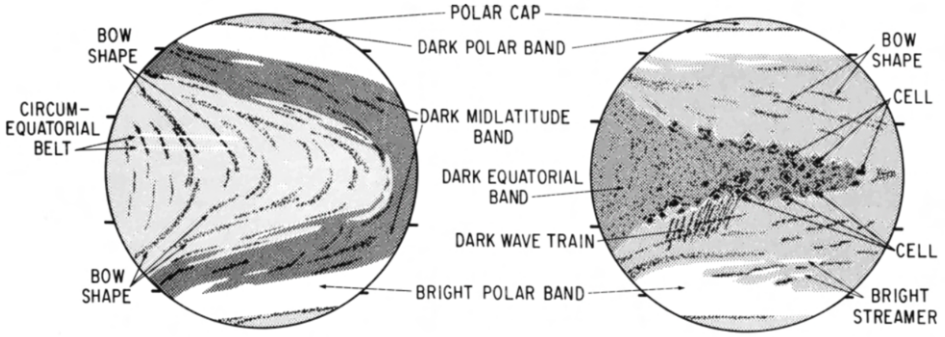

Figure 5: Sketch of Venus' atmosphere and the features observable from far away from the superficial layers.

And, with more close up photos, more details can be revealed.

Close-up images of the Northern hemisphere reveal many features indicating convective activity, turbulence and waves at the cloud tops. At a spatial resolution of few kilometres the cloud top has patchy morphology with dark spots and "valleys" a few tens of kilometres in size. At high latitudes (¿ 60◦N) three types of waves: long straight features, short wave trains, and irregular wave fields—are often observed. The long waves (bottom right in figure) have wavelengths of a few tens of kilometres

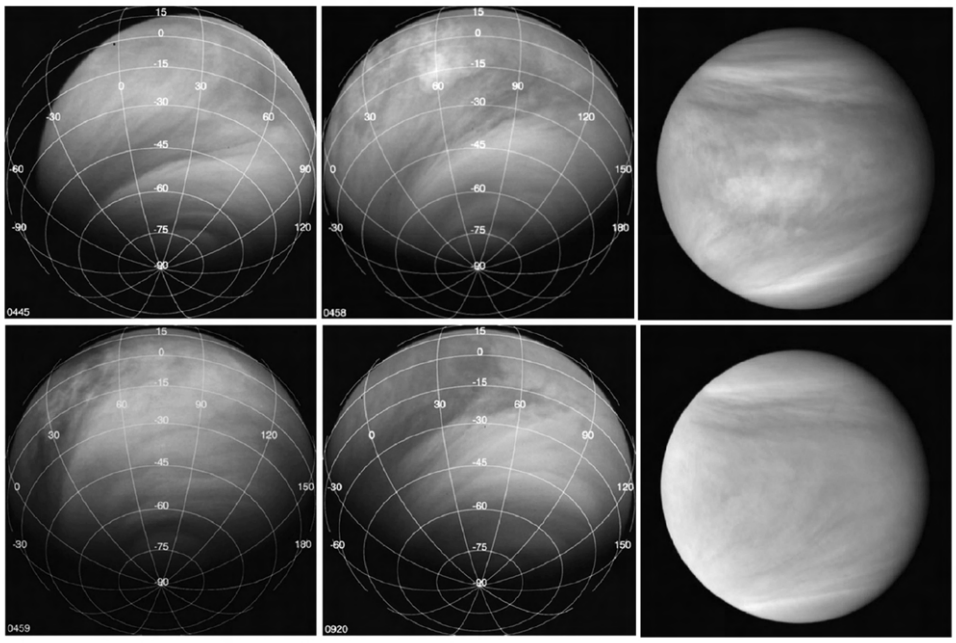

Figure 6: Close-up features of Venus' clouds.

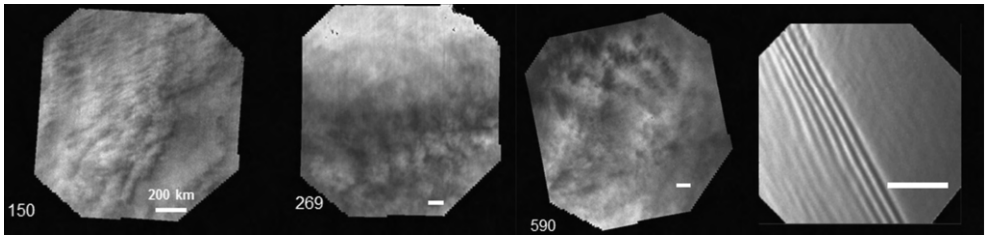

Figure 7: Other features of Venus' atmosphere.

and extend for a few hundred kilometres. Short waves form compact "trains" several tens of kilometres wide with typical wavelengths of 3–7 km.

The clouds of Venus show a great variability in appearance in the equatorial region with a nearly endless variety in shapes and sizes. Nevertheless certain regular properties can be identified even in this highly mutable part of the atmosphere. The largest structures present in the equatorial regime are the bow-like waves, which can be seen in Figure 8 and 9.

3 The data

4 Data Preparation

In this section we report the data pre-processing that has been carried out and that led us to the final dataset.

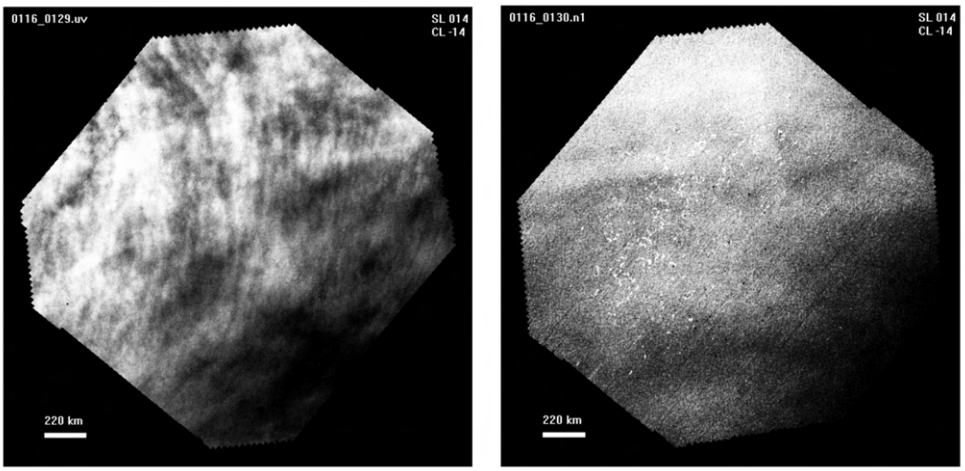

Figure 8: Details of the features of Venus' clouds.

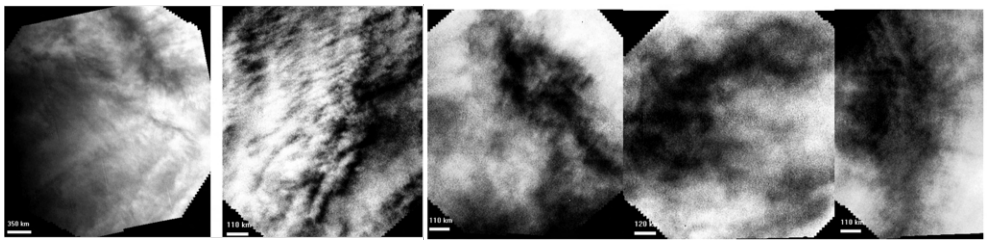

Figure 9: Further details of the features of Venus' clouds.

4.1 The datasets

The two available datasets taken into considerations are the ones provided by ESA with the Venus Express mission, available at https://www.cosmos.esa.int/web/psa/venus-express, and the one provided by the Japanese space agency with the probe Akatsuki, available at https://darts.isas.jaxa.jp/planet/project/akatsuki/. In both cases the UV camera has been selected as main instrument, since it contains the pictures that better capture atmospheric features. The two databases have been studied.

We show in Figure 10 some of the best pictures here available to show the level of detail that the dataset provides on Venus atmosphere.

Here the resolution goes from 2000-10000 km/pixel. The details of the clouds can be easily seen, and for this reason Venus Express database would have been a great choice as a viable dataset. Unfortunately, no straightforward python library has been found to read the geometry files, which are essential to be able to map all the pixels with a projection.

Akatsuki website provides seven databases of different sensors, so different images taken with different wavelenghts. The one we are interested in the most is the UV camera. Here we selected the .FIT files and the geometry files.

FITS are a standard extension for scientific images, and FITS file are divided into two parts, the head and the actual content. The header contains, for each picture, all the information regarding the metadata and geometrical information of the satellite at the moment in which the picture was taken. The second part contains the actual image. Instead, the geometry files are files where, for each pixel, a geometric information is stored. Namely, in Akatsuki geometry files, for each pixel latitude, longitude, local time, phase angle, incidence angle, emission angle, azimuthal angle are saved. Both of them can

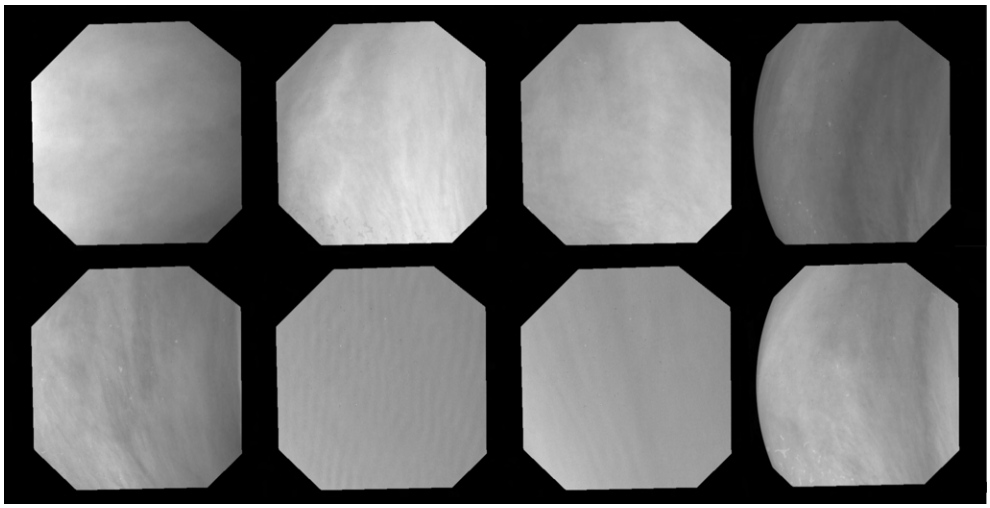

Figure 10: Pictures of Venus atmosphere taken by the ESA Venus Express mission.

be opened with a python package called *astropy*, which allow to extract the relevant information.

Here too we show some of the best and closest pictures in Figure 11.

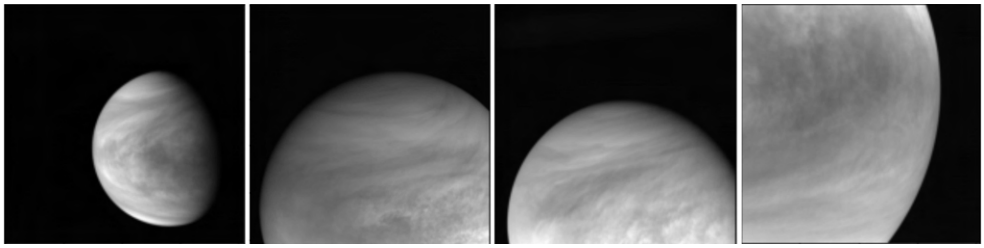

Figure 11: Pictures taken from the Akatsuki dataset.

In the end, because of the impossibility of opening the geometry file in the Venus Express database, the Akatsuki one has been selected.

4.2 Data visualisation

As previously said, the selected dataset is the one from the Akatsuki probe. The main data we used for the project has the following structure. Several pictures were taken with the UV camera, with intervals of MISSING minutes. For each picture, two .FITS file are produces. One represents the actual pic. The other one his associated geometry file, where, to each pixel, the geometrical information are associated. We show in Figure 12 different examples of the UV pic and its associated geometry information:

In order familiarize with the data and to properly understand how to preprocess it, we worked with a time interval between 2/2020 and 6/2020. During the orbiting of the probe around the planet, different altitudes are reached by the satellite. We are interested in the closest ones. To have a qualitatively idea of the range reach by the probe, we plot in Figure 13 the distances versus time.

Where the periodic resulting graph shows that the distances lie in between c.a. 20.000 km and 350.000 km. It can also be noted that the in certain orbits some data are missing, meaning that the UV camera failed to collect the pictures.

As we said, we are interested in the pictures that are as close as possible to the planet. We thus selected an arbitrary threshold distance, and we select the corresponding data. A threshold distance

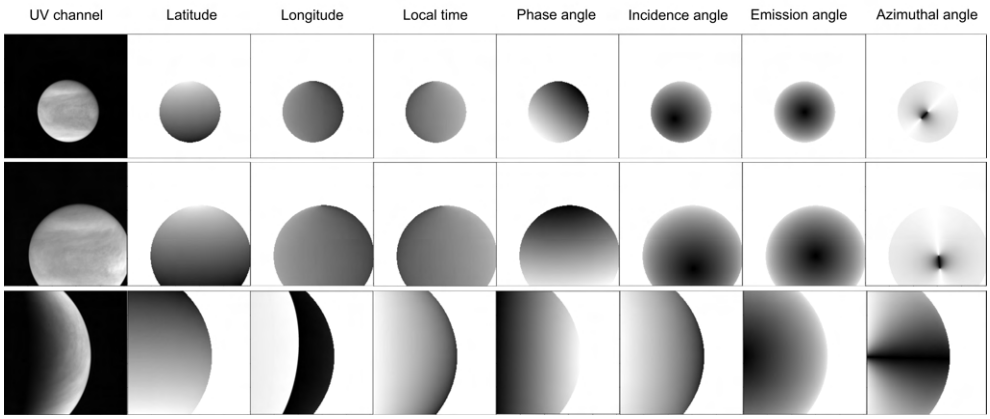

Figure 12: Geometry information of the Akatsuki dataset.

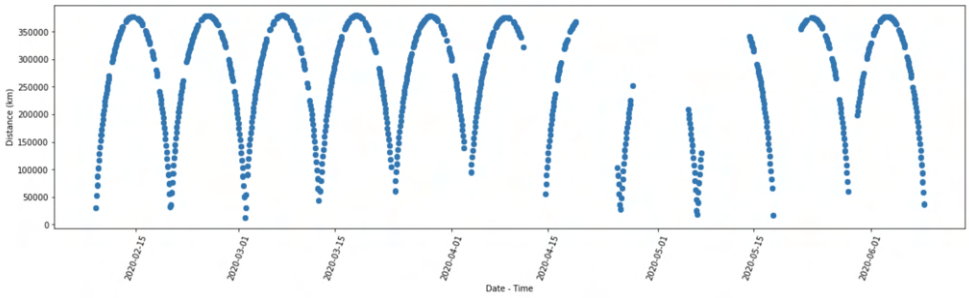

Figure 13: Graph of the altitude of the satellite with respect of the planet center during the different orbits around it.

of 200.000 km is been used. We show in Figure 14 some of the pictures collected:

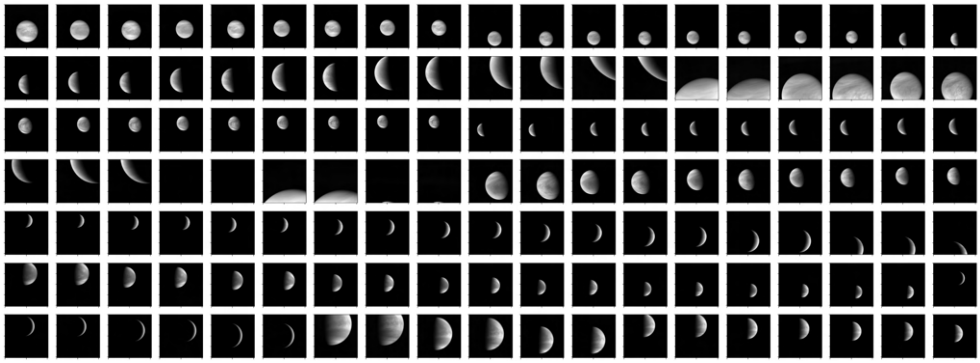

Figure 14: Samples from the dataset of the UV pictures taken from Akatsuki.

4.3 Planet cropping

Now that the data to work with has been selected, the next step is to start improving the pictures we will work with. We are obviously interested only in the part of the uv pic that actually contains information, so the gray one where contrast in between the clouds of the atmosphere are visible. Thus,

the first thing we need to do is no remove the background. The pictures have been taken while the sun was in different positions with respect to the probe. In some of them, the sun was exactly behind the satellite, thus allowing a full or almost full view of the whole planetary disc. Some examples are shown in Figure 15:

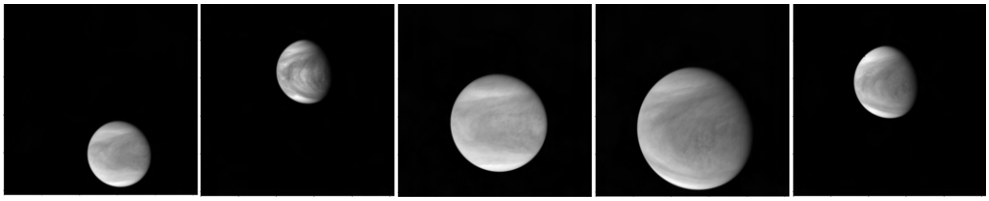

Figure 15: Pictures of the full disc of Venus taken from the Akatsuki dataset.

In other cases instead, the sun was on one side of the planet, thus creating a black shadow on the planet that does not allow us to see any atmospheric features. In Figure 16 some examples.

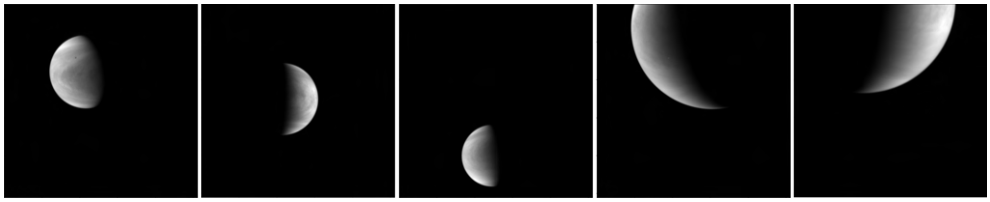

Figure 16: Pictures of Venus with the side on an angle such that the planet results mostly in shade.

Thus, apart from removing the background, one important thing to do is to crop the planet and only keep the part without the shadow. In order to do so, the geometry information has been used.

If we connect the center of the planet with, respectively, the center of the sun and a generic point of the surface, then we obtain two lines. The angle in between these two lines if called phase angle, and, as can be seen above, is one of the information that the geometry file provides.

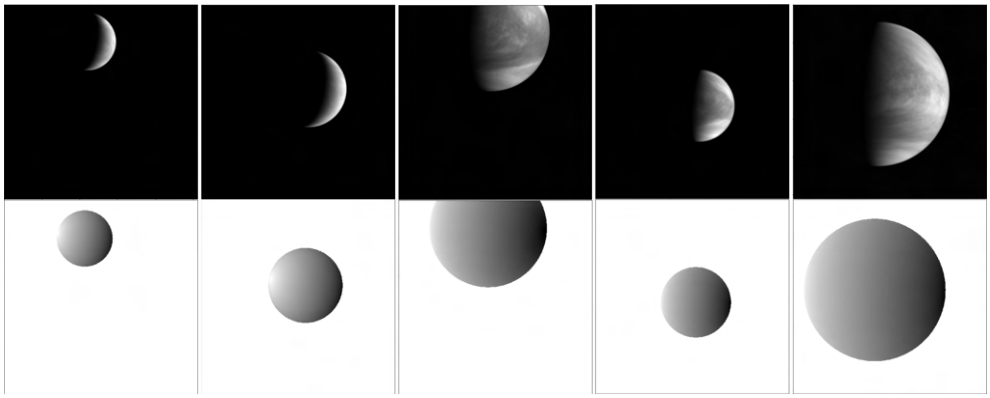

Figure 17: Comparison between the UV pictures and the geometry information related to the same picture.

In Figure 17 we show some examples of UV pictures where the probe is at a different angle with respect to the sun, and its associated geometry file with the phase angle information. The phase angle goes from 0 degrees, corresponding to black, to 180 degrees, corresponding to white. From the

geometry file it can be easily seen the direction where the sun can be found, namely is the one where the pixels get darker.

Geometrically it's easy to understand that the portion of the planet (and correspondingly UV picture) we are interested in is the one where the phase angle goes from 0 to 90 degrees. In reality, a threshold angle of ca. 85 degrees has been selected. In conclusion, all the pixels where the phase angle is greater than 85 degrees (the shaded area), or where there is no geometric information (the background) have been selected, and the same have been removed from the UV picture, as is shown in Figure 18.

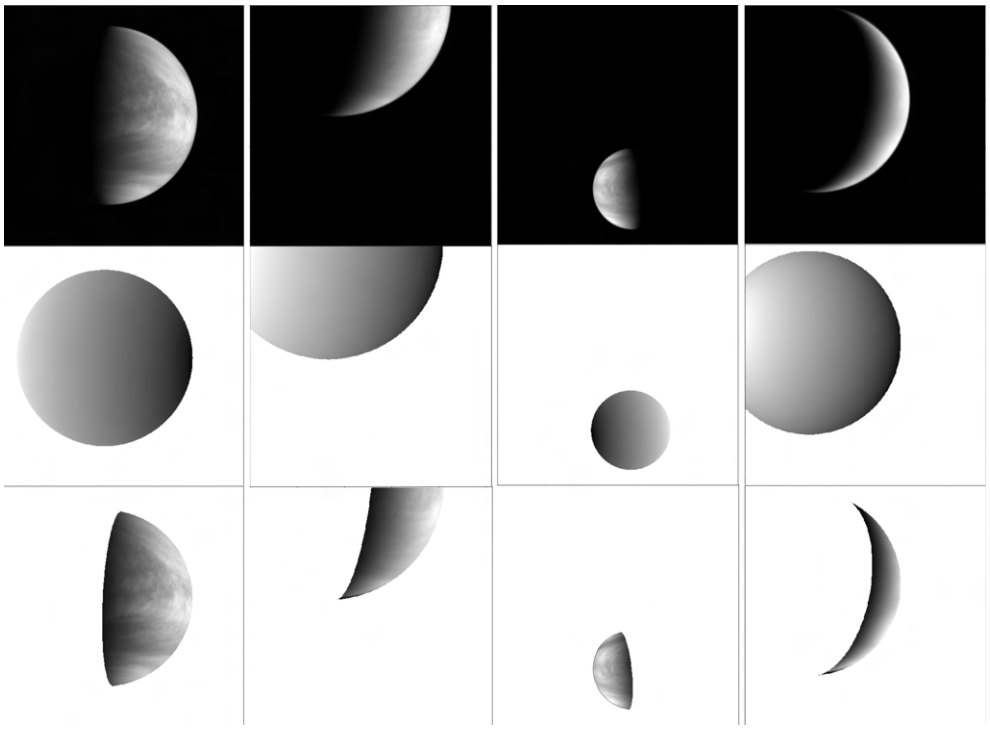

Figure 18: Cropping of the UV picture on the basis of the corresponding geometry file.

4.4 Normalization with respect to the phase angle

The next important step in the analysis of the cropped picture of the planet is their intensity normalization with respect to the phase angle.

The part of the planet more directly exposed to the sun, so the one with lower phase angles, benefit from a higher brightness. This brightness slowly fades into the the darkness of the not exposed part of the planet, where the phase angles reach values above 90 degrees.

In order to study pictures of Venus to understand the features and details of the atmosphere, we would prefer to have the whole picture with the same level of brightness exposure. For this reason we want to enhance the brightness of the pixels of the picture depending of their phase angle. The pixels corresponding to a higher phase angel, so closer to the dark area, need to have their values increased in such a way that the final brightness feels homogeneous in the whole exposed face of the planet.

The way this issue has been addressed, was by plotting on a graph all the pixel intensities versus the cosine of their respective phase angle. Then, the intensities have been fit with a line, and every pixel intensity has been normalized with respect to the fitted line, as shown in Figure 19.

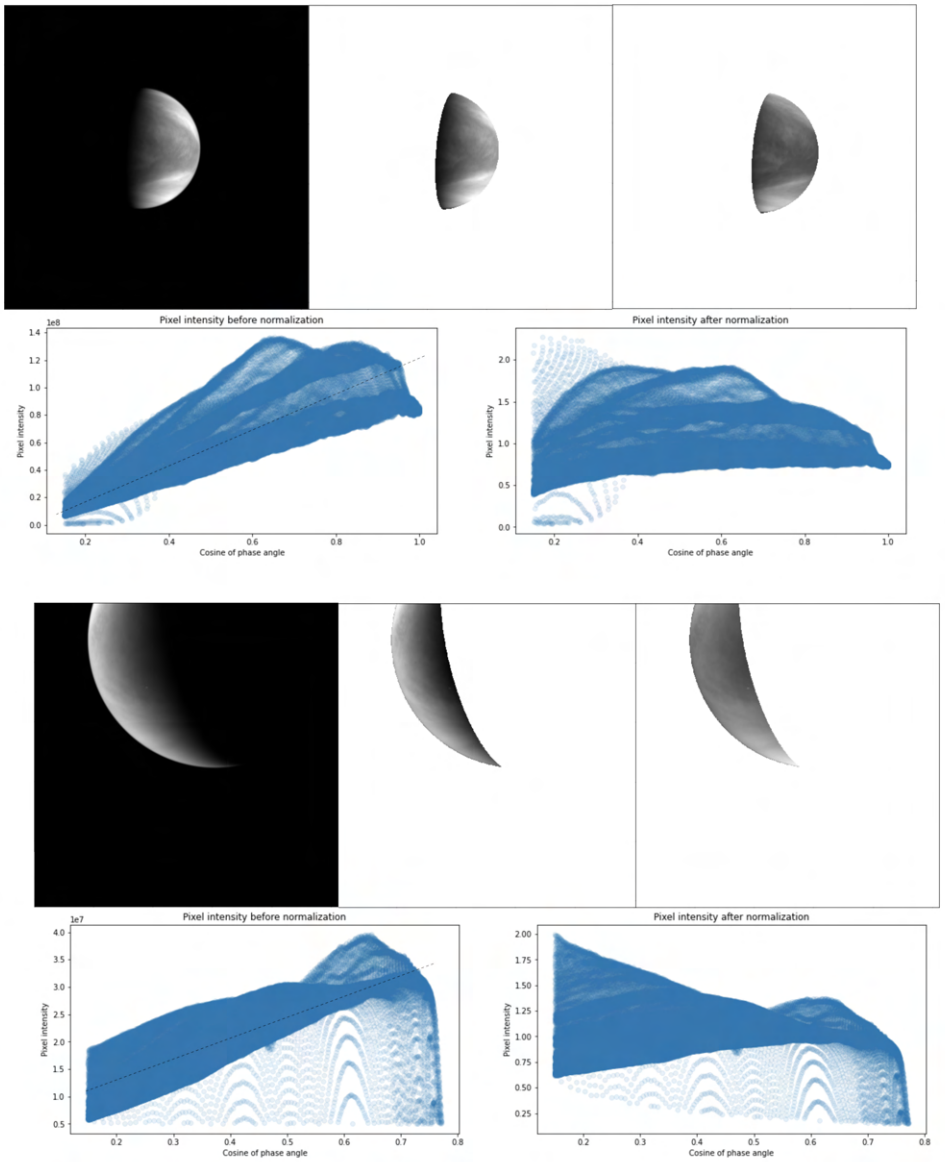

Figure 19: Normalization of the cropped pictures with depending on the phase angle.

4.5 Projection

Once the pictures have been properly cropped and normalised with respect to the phase angle, the next thing to do was to choose an appropriate projection in order to represent the venus surface in a way that would evenly spread the information in every pixel, removing the fact that in this way the information contained in every picture is being affected from the curvature.

There are several available projections. To give some examples in Figure 20 some of the most common ones are shown.

The mapping has been performed through some of the most common python packages. In particular, in this case *basemap* has been used. The projection selected is one of those shown above, in particular the "Cylindrical Equal-Area Projection". This has been chosen since its projected in such

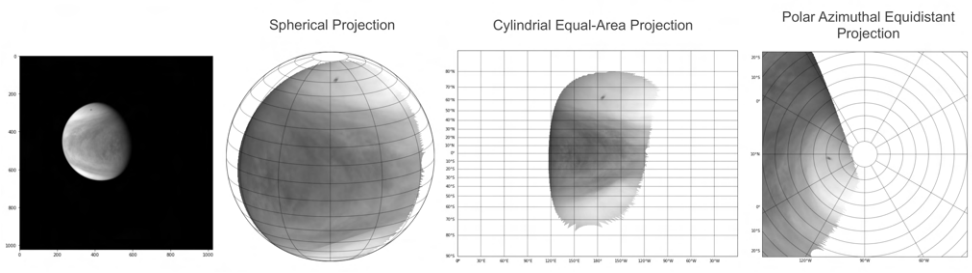

Figure 20: Normalization of the cropped pictures with depending on the phase angle.

a way that the areas are comparable, which is what we want if we are supposed to feed the spread information of the planet surface into a neural network, which should not be affected by the curvature effect that could instead lower or higher the importance of certain parts of the picture depending on where they are located on the curvature of the planet.

In Figure 21 are examples of projected pictures are shown.

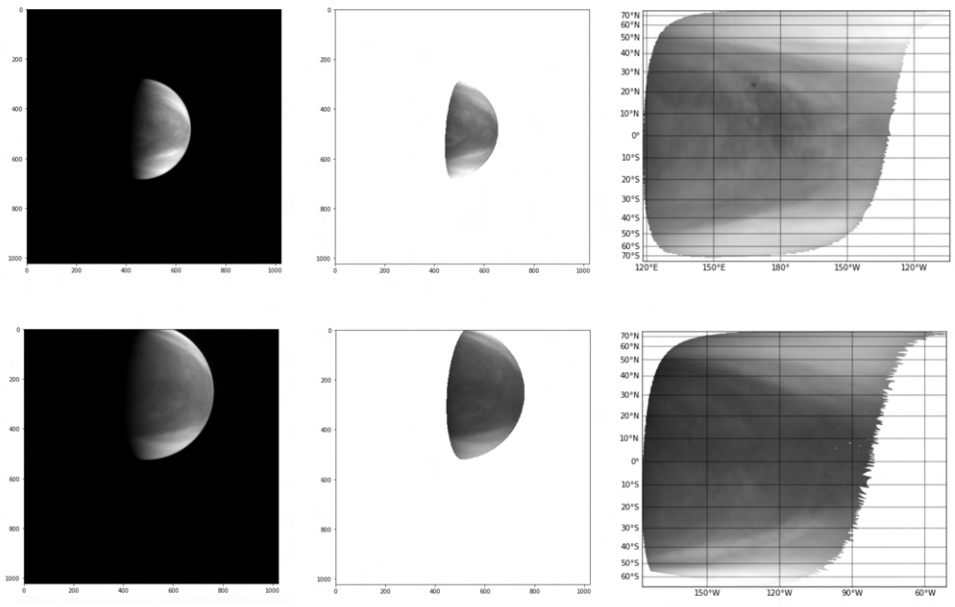

Figure 21: Examples of some of the possible projections.

Where all the projections have been cut in order to only keep the part of interest, namely the one where the limits of the latitudes and longitudes are reached. Also parallels and meridians have been drawn on top just to show how the mapping has been made, but have been of course removed afterwards.

One last thing to notice is the fact that the projection used showed some problems with some geometry files. The issue is that when assigning to every pixel a value for the longitude, ranging from 0 to 360, it can happen that the satellite is taking a picture from an angle where the longitude is not continuous, since the cut between 360 and 0 degrees is visible. In this case the projection would show part of the planet on one side of the map, and the other part on the opposite side. These two parts are joined by strained lines due to how the package fills the empty spaces, as is visible in Figure 22.

In the central picture, representing the longitudes of each pixel of the planet, it can be seen the

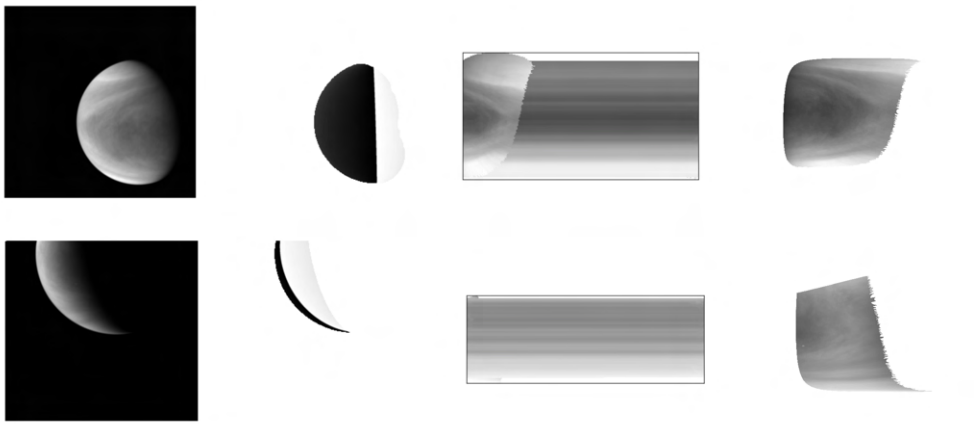

Figure 22: Examples of the issue caused by the presence of very high and low longitudes in the geometry file and their shifting and thus correction of the projection.

black part, corresponding to high values of longitudes, so close to 360, and the white one, with pixels whose longitudes is close to 0.

In order to fix this, the all the longitudes values have been shifted of 180 degrees, in order to center in the projected map the surface of the planet. We stress the fact that is such a way we loose the information on where the pixel is located with respect to the latitude and longitude of the planet, but this is not important since Venus atmosphere rotates way faster than the planet, as mentioned before, so that the same cloud would anyway appear in more points of the planet during the (Venus) day.

4.6 Cropping of the image

The projected pictures, once removed the parallel, meridians, and grids, are almost ready to be used as inputs of the neural networks used for the classification task. The last thing to do is to crop square pictures out of it.

I proceeded step by step. First of all, trying to crop the biggest square possible out of every single projected picture.

In order to do that an algorithm has been written to find the biggest square of ones in a matrix out of zeros and ones. In our projected pictures, the zero is the background, while every pixel with an intensity grater than zero would be considered as a possible one.

Some examples are shown in Figure 23.

Later we tried to cut smaller pictures for every projection, depending on the side length of the square.

Some examples in Figure 24

The next step to make things more precise is to actually select the square keeping into account the fact that the satellite is at different distances with respect to the planet. This means that the cropped squares in the previous picture are not actually comparable in size to other squares cropped from other pictures when the satellite was further away or closer.

In order to take this effect into account the proper way would be to also project the geometry information with the same type of projection. In this way, for each pixel in the projected picture, we can know to which latitude and longitude it corresponds. Alternatively each point could be inversely projected to go back to the original picture, and see which latitude and longitude it has.

We chose the first method and, in such a way, when the algorithm to crop the pictures is given a width, it can actually crop different squares whose dimensions correspond to a given latitude and longitude extension.

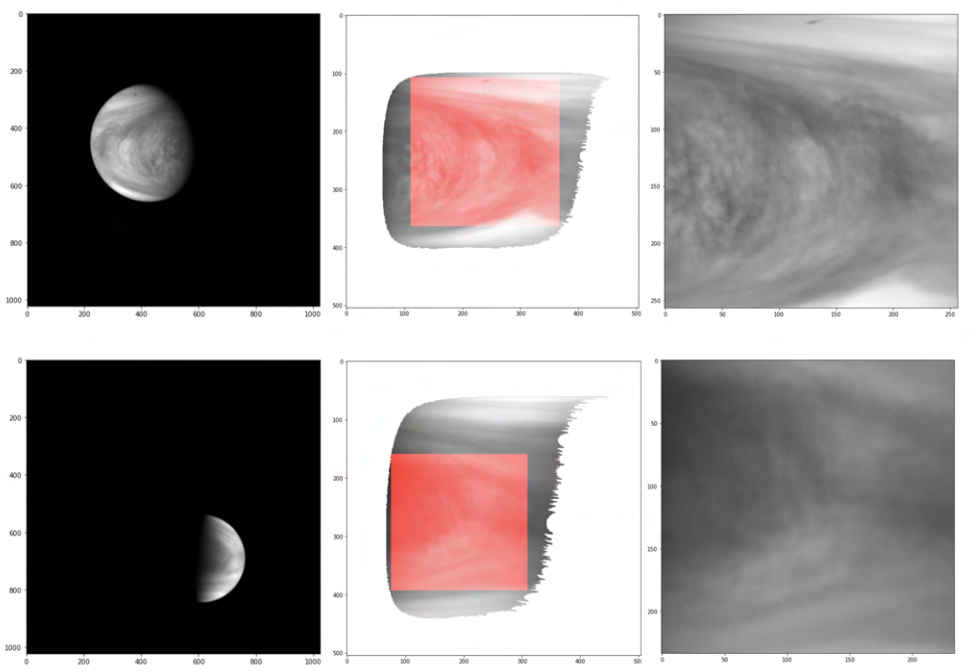

Figure 23: Biggest possible square cut inside the projection of the UV picture, once this has been preprocessed.

Some examples of the projection of both UV information and geometry file are shown in Figure 25.

Although the method is supposed to be working and should lead to actually comparable squares where the surface areas are really the same, is has not been carried out due to technical and time problems. In fact, the library used allows only to plot the projections, without being able to extract it somehow as a matrix containing the values. In fact, the projections correspond to a field where the latitude and longitude values are continuous. Trying to extract this information with a matrix (which we should need to crop the pixels) means to loose a lot of information turning the continuous values of the latitudes and longitudes into discrete steps. This would means to select a way to average down all the geometry information that would belong to the same pixel. In conclusion, a procedure which did not seem worth to go thorough considering the time limits and scope of the project, but which was surely worth to mention to report what we think is the correct way to proceed to carry out more properly the analysis.

In the end, the multiple crops technique has been used, selecting an arbitrary length of the sides of the squares.

4.7 Geometry file correction

Here we describe an attempt made to solve a problem which in the end turned out to be not necessary. The techniques used have been though interesting and instructing, and it seems thus worth analyzing them.

The problem we aimed to solved is the fact that the geometry files did not, in many pictures, match the uv files. By that I mean that the pixels represented the planetary disc where many times shifted, so that, in the procedure of removing in the planet cropping chapter, the part of the picture cropped would be not exactly only the background and the part of the planet that was too dark to be of any use. Since the geometry and UV file where not overlapping perfectly, part of the visible planet would be cropped and part of the background left, as shown in Figure 26.

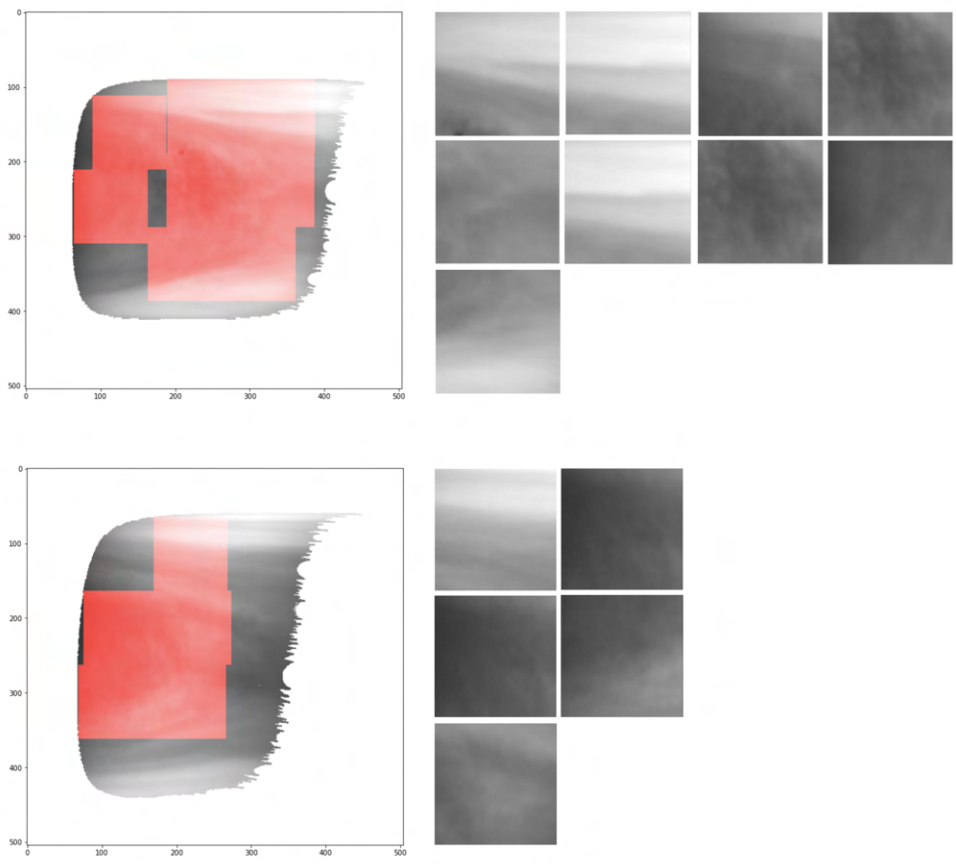

Figure 24: Multiple squares cut inside the projection of the UV picture, once this has been preprocessed.

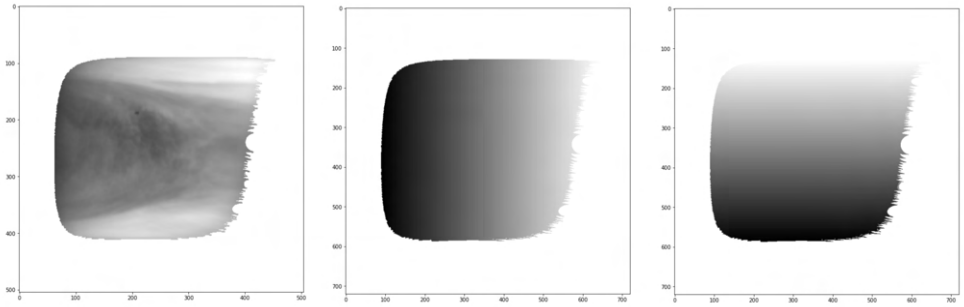

Figure 25: Examples of the UV pictures projected according to the chosen atlas.

In order to fix this mismatch, what we tried to do was to recenter the UV files. Namely we tried to perform an edge detection of the UV and geometry file. Once the edge of the planet was detected, the point forming the edge have been been fitted with a circumference.

One example of the procedure applied to the UV and geometry file is shown in Figure 27.

Once the fit has been performed, the radius and the x and y of the center have been calculated. These values have been compared between the two types of picture supposed to represent the planet at the same instant. The results are shown in Figure 28.

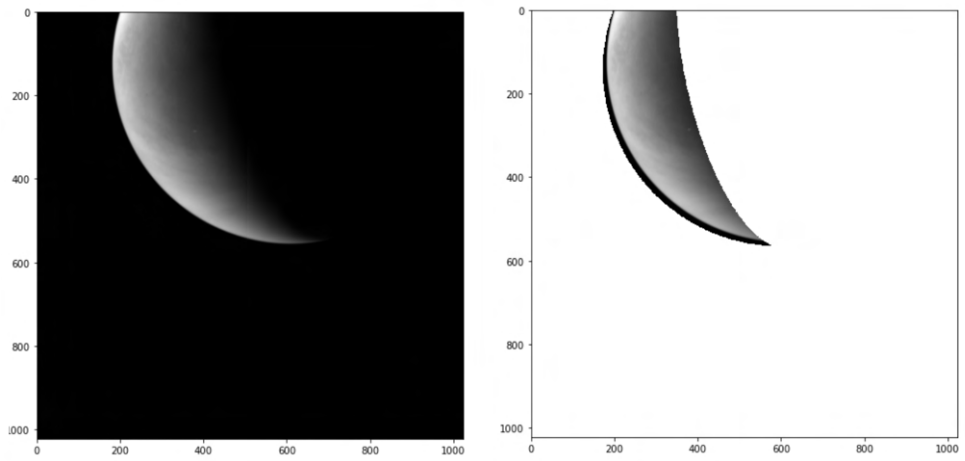

Figure 26: Examples of the geometry file shifted from the UV picture.

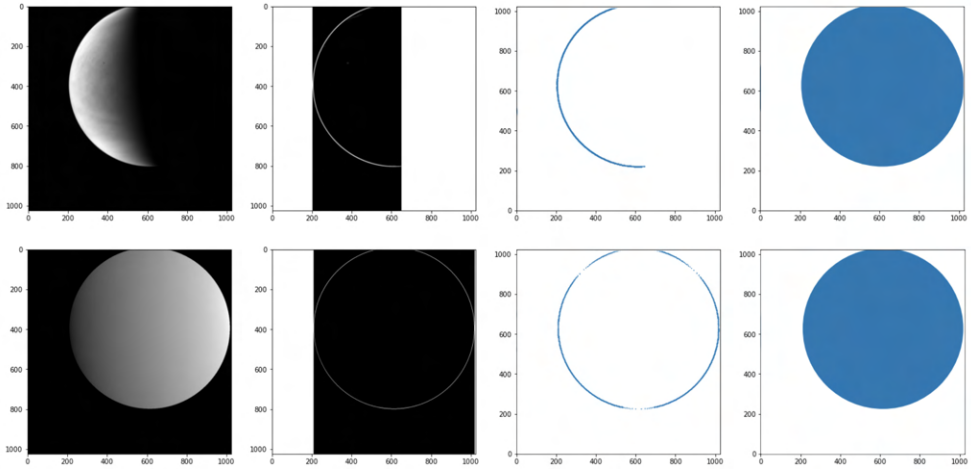

Figure 27: Edge detection procedure.

We would expect the ratio of the radius to be 1 and the x and y shifts to be 0. It's easy to see that they are far away from those values. The values have been then fitted with a line to find the average shift of the pixels in the x direction and in the y direction, and this value is been used to correct the pictures, shifting all the pixels of these values in the corresponding axis. The result was definitely useful, but as mentioned, in the end the corrected dataset has been found and used instead.

5 Machine Learning techniques for image classification

Once having a procedure to turn the satellite images into a dataset of cropped square pictures, the next step has been to develop a technique to classify such images. We set the Venus images aside and we worked instead with ready dataset, in order to validate the best method between the ones available on a dataset that had already been tested.

Online there are several clouds classification datasets, but, in particular, the *SWIMCAT* (Singapore Whole sky IMaging CATegories Database) dataset has been used.

Proceding step by step, we first started with supervised classification, to continue then with unsu-

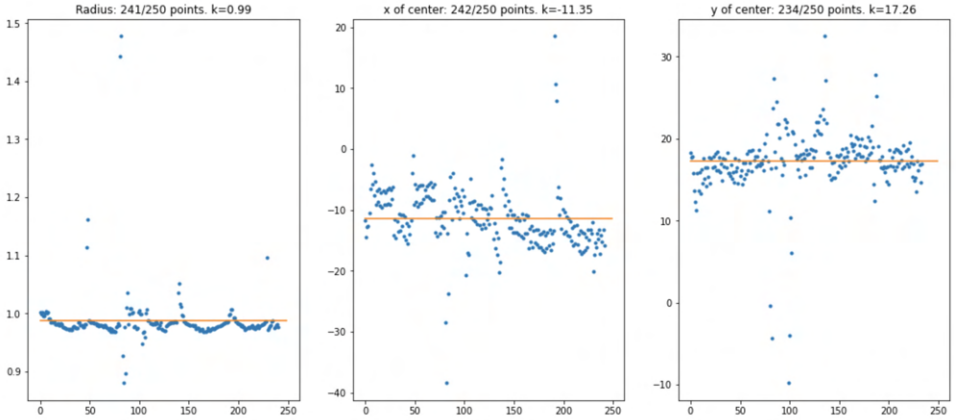

Figure 28: Plot of the the differences between the x, y coordinates of the original center and the estimated one, and the differences in the radius.

pervised classification, which is eventually the true goal of the project, since we don't have any label assigned to the cropped pictures.

5.1 Supervised

5.1.1 Convolutional Neural Network

As said, the first technique used to classify was a supervised approach with a ready dataset. We present in Figure 29 some of the types of clouds present in the SWIMCAT dataset.

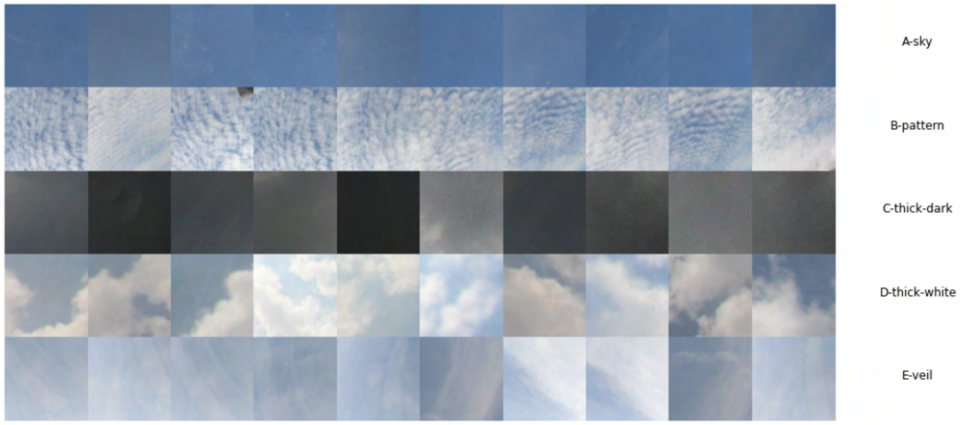

Figure 29: Visualization of the Swimcat dataset.

The dataset is unbalanced, meaning that different classes have a different number of pictures. The class with less elements has been found, and its number of elements has been set as the number of pictures to pick up from every class. After some classical preprocessing, as, for example, normalization, a simple convolutional neural network has been created using Keras with the model in Figure 30

The model has then been trained and tested on the dataset. In Figure 31 we present the results: accuracy and loss for both training and validation, and the confusion matrix:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 125, 125, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 62, 62, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 62, 62, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 31, 31, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 31, 31, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 31, 31, 16)        4624
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 15, 15, 16)        0
_________________________________________________________________
flatten (Flatten)            (None, 3600)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 3600)              0
_________________________________________________________________
dense (Dense)                (None, 32)                115232
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 165
=================================================================
Total params: 130,165
Trainable params: 130,165
Non-trainable params: 0
_________________________________________________________________
```

Figure 30: CNN model used for the supervised classification.

5.2 Unsupervised

The next step has been to try unsupervised learning techniques to learn how to autonomously divide the pictures into different classes. We first tried with different classical approaches, where by classical we mean that no use of neural networks was made. So techniques like k-means, PCA, t-SNE, where used. Later on we proceeded with deep learning approaches, like autoencoders or pre-trained convolutional neural networks.

5.2.1 K-means

As a first direct attempt, the k-means algorithms has been used to try to divide the pictures into classes. At first the data has been normalised, but all these was not sufficient to obtain any good result. Referring to the dataset showed at the beginning of the dataset, pictures from class A (sky) and C (thick dark) could be easily divided, but all the other three classes where completely mixed up.

5.2.2 PCA and t-SNE

We proceeded with more sophisticated classical approaches. Here we first performed a principal component analysis, to reduce the dimension of the picture, once turned into a vector, keeping only the most important dimensions.

Later we performed a t-distributed Stochastic Neighbor Embedding, which has the goal to make closer pictures (already reduced with PCA) that look more alike, and move away the ones that look more different. Once this was done, we plotted them in a greed keeping the proportion, in order to see visually the final result. The results are shown in Figure 32.

we then proceeded again with techniques like K-means to group and cluster the pictures. These techniques divide the elements in clusters depending of some kid of distance definition. Unfortunately, it can already be seen visually that the groups created are very distinct, once again, for the clear sky and thick dark clouds. The other classes are completely mixed up int he third cluster on the bottom left side.

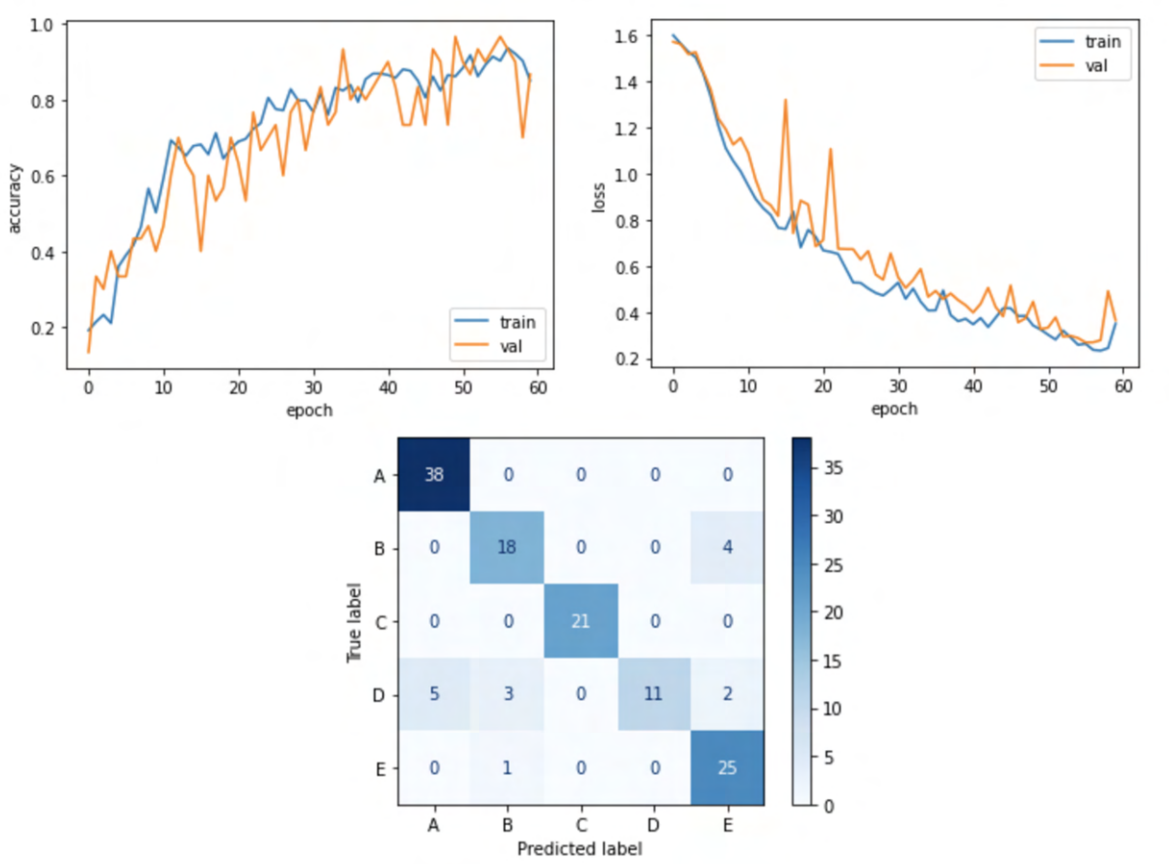

Figure 31: Metrics of the CNN training.

It's clear that the clustering depends a lot, in this case, on the color of the pictures. We would be more interested instead to learn how to divide them depending on the patterns. Thus the same procedure has been attempted also by converting the pictures into gray scale.

The result was unfortunately the same.

5.2.3 Pretrained VGG16 model

We have then tried to cluster the pictures with some models available through Keras. These models are very long and complex architectures that have been previously heavily trained to classify pictures of any king, and can then be downloaded and imported already compiled are ready to be used.

The first we used was the VGG16 model, presented in Figure 33.

the model is a classical convolutional neural network, with the only peculiarity of being very complex and thus very hard to train locally. It's been trained with a big dataset that allowed the architecture to learn how to extract properly the important features of a picture given as an input. The model ends with three fully connected layers, which means that it is capable of storing the important information of one picture into the last fully connected layers of the architecture.

The last fully connected layers, of shape $1 \times 1 \times 1000$ will be used to perform a PCA, in order to reduce the dimensions even further, and then the usual procedure of K-means has been applied to create clusters.

In particular, when importing the model, it can be chosen up to which layer one wants to load

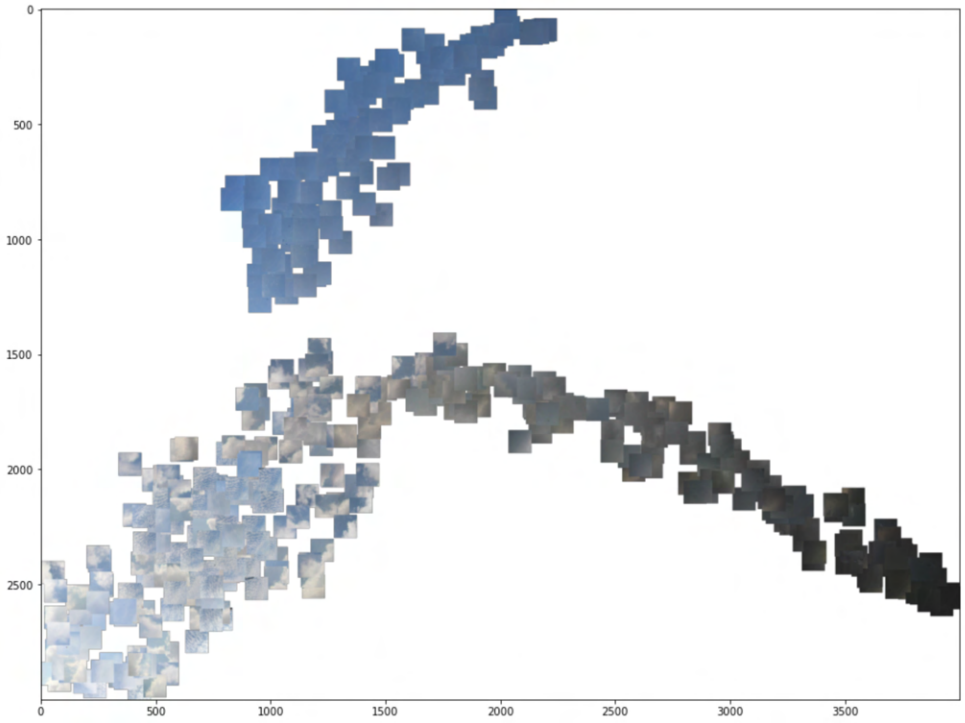

Figure 32: t-SNE algorithm applied to the Swimcat dataset.

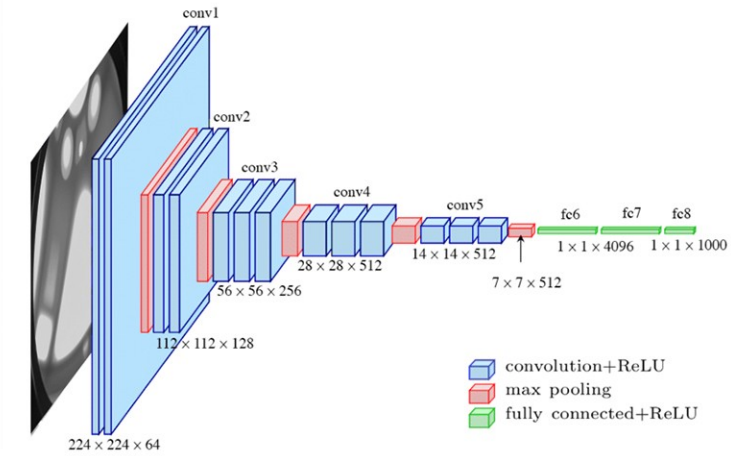

Figure 33: VGG16 model used for the unsupervised training.

it. So, the above procedure (fully connected layer, PCA, Kmeans) has been applied selecting different final layers between the last three ones.

The best results have been obtained when loading the model up to the last red layer in the picture, namely until the last convolutional step, after the flattening is applied (since we need a vector to perform the PCA anyway) but before any fully connected layer was applied.

The model is shown in Figure 34.

We also present the results obtained once reducing the vector of length $7 \times 7 \times 512 = 25088$ with

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0

Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

Figure 34: VGG16 model used for the unsupervised training cut before any fully connected layer was applied.

the PCA to 500 components and after clustering with Kmeans into 6 different clusters. Actually the cluster of the dataset we showed before are supposed to be 5, but, since one of them, the thick dark one, kept of being split into two different classes, we let the Kmeans method the freedom to divide them into two different clusters, since this in our project we don't actually know how many clusters there are going to be, and the amount of clusters could be variable anyway, as long as it is able to cluster properly them.

These are the results

Cluster n.	1	2	3	4	5	6
Clear sky	0	219	0	0	5	0
Pattern	0	0	89	0	0	0
Thick dark	127	0	0	122	0	2
Thick white	3	0	0	0	32	100
Veil	0	1	0	0	78	6

Apart from some confusion made between some thick white and veil clouds, the pictures are well clustered.

5.2.4 Pretrained DenseNet model

The same procedure has been applied with a pretrained DenseNet model, namely the DenseNet169 from Keras. In this case, we removed the last layer and extracted the features at the fully connected layer before the last one. Then, as before, PCA and KMeans have been applied.

The results are the following

Cluster n.	1	2	3	4	5	6
Clear sky	3	0	221	0	0	0
Pattern	89	0	0	0	0	0
Thick dark	100	35	0	36	44	36
Thick white	135	0	0	0	0	0
Veil	85	0	0	0	0	0

5.2.5 Autoencoder

Another technique that has been used is the autoencoder. An autoencoder is an architecture that shrinks progressively and then symmetrically expands again, as in Figure 35.

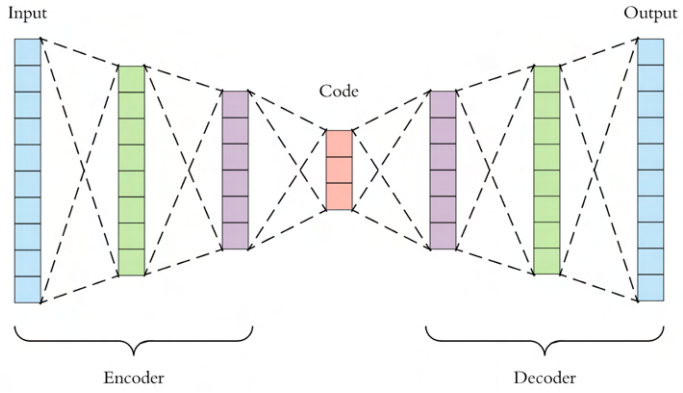

Figure 35: Conceptual representation of an autoencoder.

It is composed by two main parts, the encoder and the decoder. The goal of an autoencoder is to learn the identity function, in order to reproduce as close as possible the input received. In our case, since we work with images, the encoder is composed of convolutional layers, which downsample the image, until the information of the convoluted input is stored in a linear layer. The decoder part is instead made of upsampling layers.

The idea is that, if the identity function is properly learned, then the middle linear layer should have good information about the compressed picture. This linear vector with the essential information can go through the same process of PCA and KMeans.

In our case, different architectures have been tried. In Figure 36 we present one to give an idea of which models have been trained: on the left the encoding part, flattening the picture in its last layer, and on the right the decoding part, starting from a flat layer and expanding it back into a picture.

The the two parts have been connected so to create out autoencoder, which has been trained on the same dataset showed before, namely the SWIMCAT dataset. In Figure 37 we present the results of the train and validation loss.

In spite of the low validation and train loss, we present in Figure 38 some results of the input picture and the output one, which is supposed to be as close as possible to the input.

As it can be seen, the autoencoder is not able to properly reproduce the pictures. Party it manages to reproduce the main shapes of the inputs, but the blurred output doesn't allow to distinguish between veil clouds and thick white clouds that are blurred.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 128, 128, 3)]	0
conv2d (Conv2D)	(None, 128, 128, 16)	448
max_pooling2d (MaxPooling2D)	(None, 64, 64, 16)	0
leaky_re_lu (LeakyReLU)	(None, 64, 64, 16)	0
batch_normalization (BatchNo	(None, 64, 64, 16)	64
conv2d_1 (Conv2D)	(None, 64, 64, 32)	4640
max_pooling2d_1 (MaxPooling2	(None, 32, 32, 32)	0
leaky_re_lu_1 (LeakyReLU)	(None, 32, 32, 32)	0
batch_normalization_1 (Batch	(None, 32, 32, 32)	128
conv2d_2 (Conv2D)	(None, 32, 32, 64)	18496
max_pooling2d_2 (MaxPooling2	(None, 16, 16, 64)	0
leaky_re_lu_2 (LeakyReLU)	(None, 16, 16, 64)	0
batch_normalization_2 (Batch	(None, 16, 16, 64)	256
conv2d_3 (Conv2D)	(None, 16, 16, 128)	73856
max_pooling2d_3 (MaxPooling2	(None, 8, 8, 128)	0
leaky_re_lu_3 (LeakyReLU)	(None, 8, 8, 128)	0
batch_normalization_3 (Batch	(None, 8, 8, 128)	512
conv2d_4 (Conv2D)	(None, 8, 8, 256)	295168
max_pooling2d_4 (MaxPooling2	(None, 4, 4, 256)	0
leaky_re_lu_4 (LeakyReLU)	(None, 4, 4, 256)	0
batch_normalization_4 (Batch	(None, 4, 4, 256)	1024
conv2d_5 (Conv2D)	(None, 4, 4, 512)	1180160
max_pooling2d_5 (MaxPooling2	(None, 2, 2, 512)	0
leaky_re_lu_5 (LeakyReLU)	(None, 2, 2, 512)	0
batch_normalization_5 (Batch	(None, 2, 2, 512)	2048
flatten (Flatten)	(None, 2048)	0

Total params: 1,576,800
Trainable params: 1,574,784
Non-trainable params: 2,016

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 2048)]	0
reshape (Reshape)	(None, 2, 2, 512)	0
conv2d_transpose (Conv2DTran	(None, 2, 2, 512)	2359808
up_sampling2d (UpSampling2D)	(None, 4, 4, 512)	0
leaky_re_lu_6 (LeakyReLU)	(None, 4, 4, 512)	0
batch_normalization_6 (Batch	(None, 4, 4, 512)	2048
conv2d_transpose_1 (Conv2DTr	(None, 4, 4, 256)	1179904
up_sampling2d_1 (UpSampling2	(None, 8, 8, 256)	0
leaky_re_lu_7 (LeakyReLU)	(None, 8, 8, 256)	0
batch_normalization_7 (Batch	(None, 8, 8, 256)	1024
conv2d_transpose_2 (Conv2DTr	(None, 8, 8, 128)	295040
up_sampling2d_2 (UpSampling2	(None, 16, 16, 128)	0
leaky_re_lu_8 (LeakyReLU)	(None, 16, 16, 128)	0
batch_normalization_8 (Batch	(None, 16, 16, 128)	512
conv2d_transpose_3 (Conv2DTr	(None, 16, 16, 64)	73792
up_sampling2d_3 (UpSampling2	(None, 32, 32, 64)	0
leaky_re_lu_9 (LeakyReLU)	(None, 32, 32, 64)	0
batch_normalization_9 (Batch	(None, 32, 32, 64)	256
conv2d_transpose_4 (Conv2DTr	(None, 32, 32, 32)	18464
up_sampling2d_4 (UpSampling2	(None, 64, 64, 32)	0
leaky_re_lu_10 (LeakyReLU)	(None, 64, 64, 32)	0
batch_normalization_10 (Batc	(None, 64, 64, 32)	128
conv2d_transpose_5 (Conv2DTr	(None, 64, 64, 16)	4624
up_sampling2d_5 (UpSampling2	(None, 128, 128, 16)	0
leaky_re_lu_11 (LeakyReLU)	(None, 128, 128, 16)	0
batch_normalization_11 (Batc	(None, 128, 128, 16)	64
conv2d_transpose_6 (Conv2DTr	(None, 128, 128, 3)	435
activation (Activation)	(None, 128, 128, 3)	0

Total params: 3,936,099
Trainable params: 3,934,083
Non-trainable params: 2,016

Figure 36: Architecture of the actual autoencoder used.

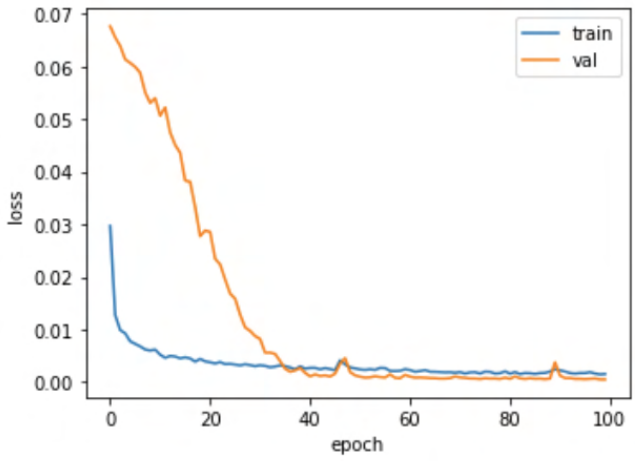

Figure 37: Loss metric of the autoencoder trained with the dataset.

This failure can also be seen with the results.

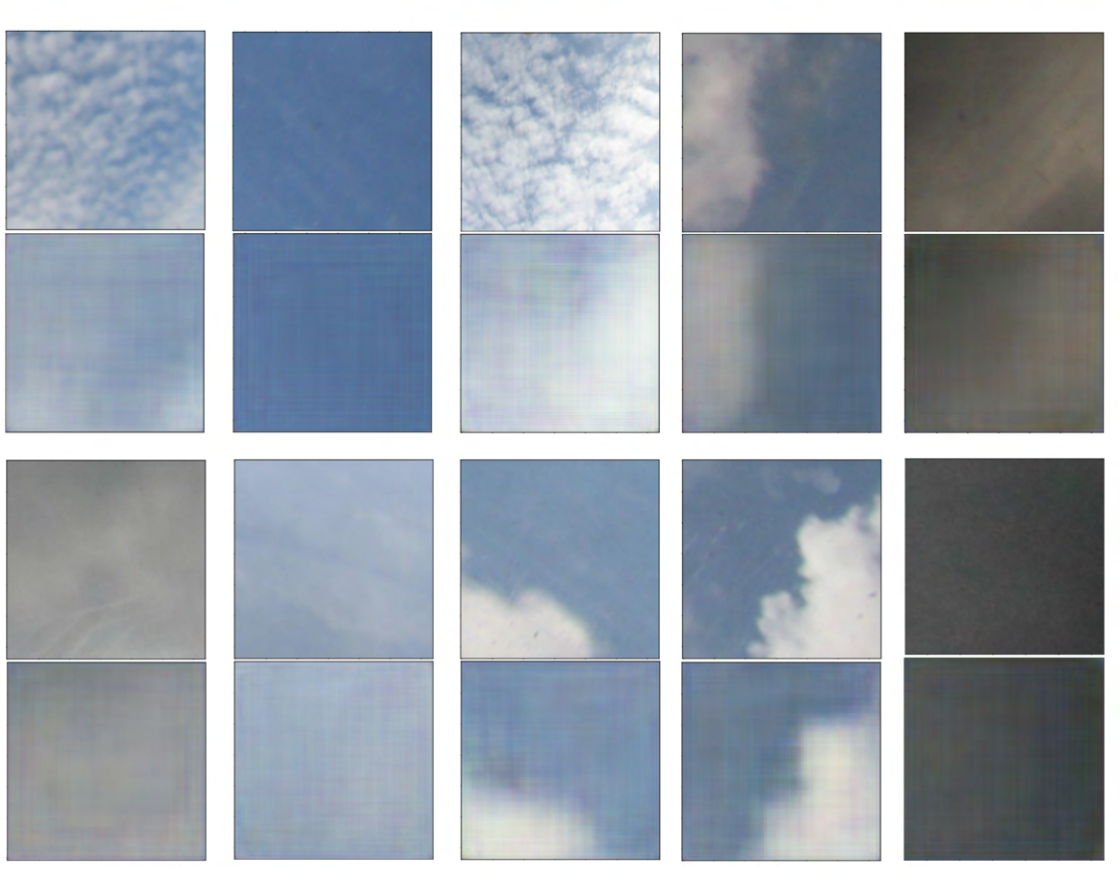

Figure 38: Results of the autoencoder when applied to the selected dataset. In the pictures, pair of pics can be seen: on top, the original one, on the bottom the one obtained as output from the autoencoder.

Cluster n.	1	2	3	4	5	6
Clear sky	0	246	0	1	0	4
Pattern	28	0	0	0	0	223
Thick dark	0	0	91	85	69	6
Thick white	65	0	0	75	0	111
Veil	124	1	0	23	0	103

The cluster are mixed, since, apparently, the autoencoder doesn't manage to encode the important features in the main flat layer.

5.2.6 Google scraper

We supposed that one of the reasons why the autoencoder was not performing as expected was the limited amount of training data. The SWIMCAT dataset is not very numerous. Thus, one possible solution was trying to train the model on a wider dataset. Other publicly available datasets have been searched and used, with similar results.

Instead of simply looking for a new dataset available online, also for the sake of learning a new methodology and technique, we decided to download the available images of clouds from google images.

A web scraper has been coded to do the task, and once again an autoencoder architecture has been trained, as shown in Figure 39.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 128, 128, 3)]	0
conv2d (Conv2D)	(None, 64, 64, 16)	1216
max_pooling2d (MaxPooling2D)	(None, 32, 32, 16)	0
leaky_re_lu (LeakyReLU)	(None, 32, 32, 16)	0
batch_normalization (BatchNo	(None, 32, 32, 16)	64
conv2d_1 (Conv2D)	(None, 16, 16, 32)	12832
max_pooling2d_1 (MaxPooling2	(None, 8, 8, 32)	0
leaky_re_lu_1 (LeakyReLU)	(None, 8, 8, 32)	0
batch_normalization_1 (Batch	(None, 8, 8, 32)	128
conv2d_2 (Conv2D)	(None, 4, 4, 64)	51264
max_pooling2d_2 (MaxPooling2	(None, 2, 2, 64)	0
leaky_re_lu_2 (LeakyReLU)	(None, 2, 2, 64)	0
batch_normalization_2 (Batch	(None, 2, 2, 64)	256
flatten (Flatten)	(None, 256)	0

Total params: 65,760
Trainable params: 65,536
Non-trainable params: 224

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 256)]	0
reshape (Reshape)	(None, 2, 2, 64)	0
conv2d_transpose (Conv2DTran	(None, 4, 4, 64)	102464
up_sampling2d (UpSampling2D)	(None, 8, 8, 64)	0
leaky_re_lu_3 (LeakyReLU)	(None, 8, 8, 64)	0
batch_normalization_3 (Batch	(None, 8, 8, 64)	256
conv2d_transpose_1 (Conv2DTr	(None, 16, 16, 32)	51232
up_sampling2d_1 (UpSampling2	(None, 32, 32, 32)	0
leaky_re_lu_4 (LeakyReLU)	(None, 32, 32, 32)	0
batch_normalization_4 (Batch	(None, 32, 32, 32)	128
conv2d_transpose_2 (Conv2DTr	(None, 64, 64, 16)	12816
up_sampling2d_2 (UpSampling2	(None, 128, 128, 16)	0
leaky_re_lu_5 (LeakyReLU)	(None, 128, 128, 16)	0
batch_normalization_5 (Batch	(None, 128, 128, 16)	64
conv2d_transpose_3 (Conv2DTr	(None, 128, 128, 3)	1203
activation (Activation)	(None, 128, 128, 3)	0

Total params: 168,163
Trainable params: 167,939
Non-trainable params: 224

Figure 39: Autoencoder architecture trained with the dataset from the google scraper.

We present the training and validation losses in Figure 40.

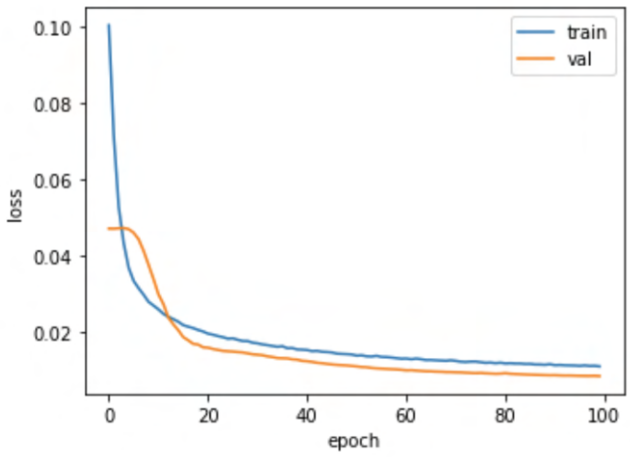

Figure 40: Metrics of the autoencoder trained on the google dataset.

In spite of the training and validation loss decreasing, looking at encoding and decoding of some pictures is enough to see how the autoencoder is not working as expected, as is shown in Figure 41

trying the unsupervised classification with 4 clusters, the results are obviously not good, since, once again, the autoencoder learn only how to properly reproduce the general colors and characteristics of the pictures, but not the details.

We present the table of results, which has been made in the following way: with the web scraper all the clouds corresponding to some keywords have been download. The keywords being "cirrus clouds", "cumulus clouds", "nimbostratus clouds" and "cirrocumulus clouds".

All the pictures downloaded from the same keyword are considered originally belonging to the same cluster.

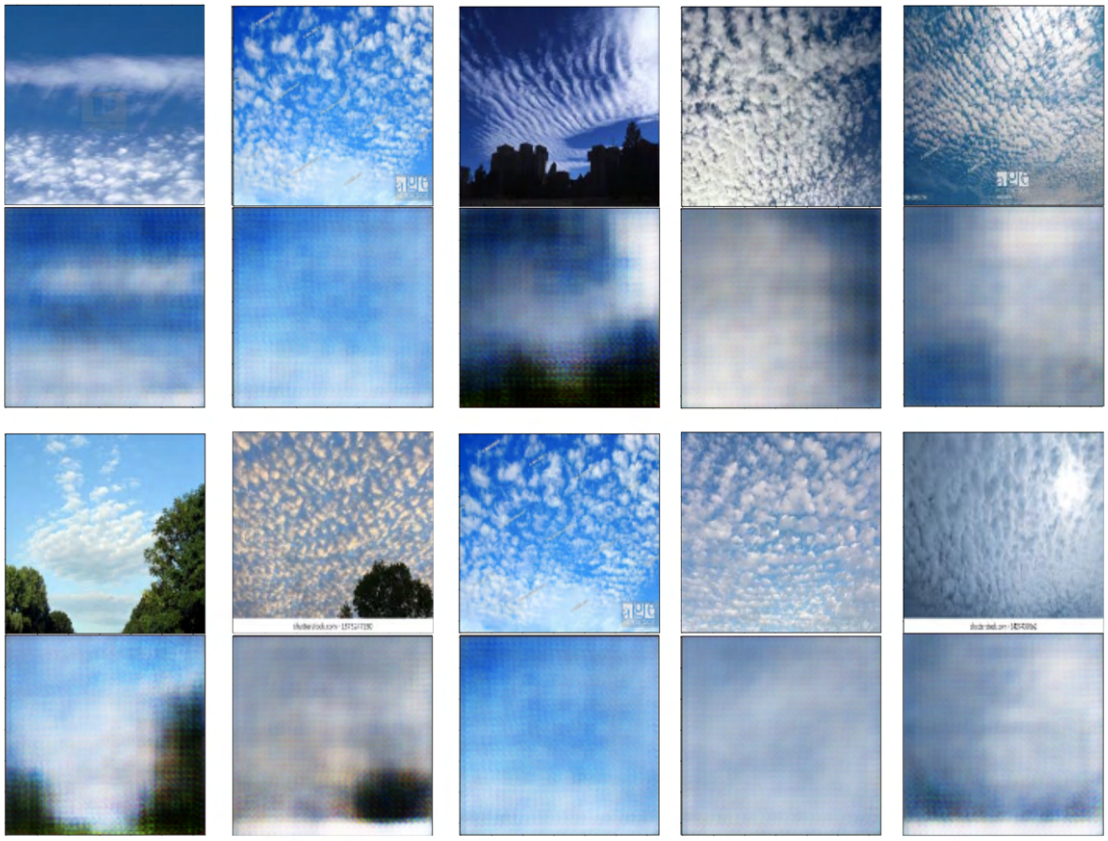

Figure 41: Results of the autoencoder on the google dataset. Again, on top the original picture while on the bottom the output one.

The autoencoder with a following PCA and KMeans classified in the following way the pictures.

Cluster n.	1	2	3	4
cirrus	48	196	100	9
nimbostratus	105	71	47	71
cirrocumulus	35	5	49	82

5.2.7 Conclusions

In conclusion, we observe that the VGG16 model is the one who can better autonomously cluster the pictures, and this is the one selected to proceed.

The other methods have been tried with different combinations of datasets and hyperparameters, but with no better results than the ones shown here.

6 Results

We apply the described procedure to the dataset of the Akatsuki satellite. The complete algorithm is shown in Algorithm 1.

The values used for the inputs are the threshold distance of $\delta = 200000$km, the number of clusters $N = 9$, the side of the cropped squares 120 pixels. The results obtained are shown in Figure 42, where, for each cluster, a sample of random pictures has been shown.

It can be seen how the algorithm successfully learns how to cluster the cropped pictures depending on their features. The first important thing to point out, is that the chosen dataset doesn't show the atmosphere of Venus close enough to distinguish more features that the ones visible on the results. The algorithm can anyway be tested considering the current dataset as a proof of concept. In fact, as it can be seen, the main differences are distinguished: clouds with horizontal-lines patterns, or tilted lines. This can be seen in cluster 1, 3, 4, 5 and 8. Even though all of these clusters have the same type of clouds-pattern, the different direction of the striped is understood by the algorithm and differentiated.

Clusters 2 and 9 could have be put together, since they both show a more random-like pattern of the clouds, where no sharp line-contrast can be noticed.

Clusters 6 and 7 have probably been selected due to their darker colors. Here a not very well defined dark veil can be seen, even though no other feature is distinguishable.

In conclusion, we claim that, for the current dataset, less clusters would have been enough to highlight the main features that distinguish the cropped final pictures. Nonetheless, the algorithm should be tested on a dataset where the pictures have been taken closer to the atmosphere, and the features can be better distinguished.

Algorithm 1: Image processing main loop

```
Input: Threshold distance δ
The number of clusters N
The side l of the square to crop for the final pics
begin
    Load the data
    Select all the pictures with a distance < δ
    for every selected picture do
        Crop the part of the UV picture that has no geometrical information
        Fit the data and normalise with respect to the phase angle
        Project the UV picture in the map
        Cut multiple squares out of the projected picture
    end
    Load the VGG16 pre-trained model
    for every cropped square do
        Process the picture with VGG16 and extract the vector
        Extract the most important features with PCA
        Cluster the pics with K-Means
    end
end
```

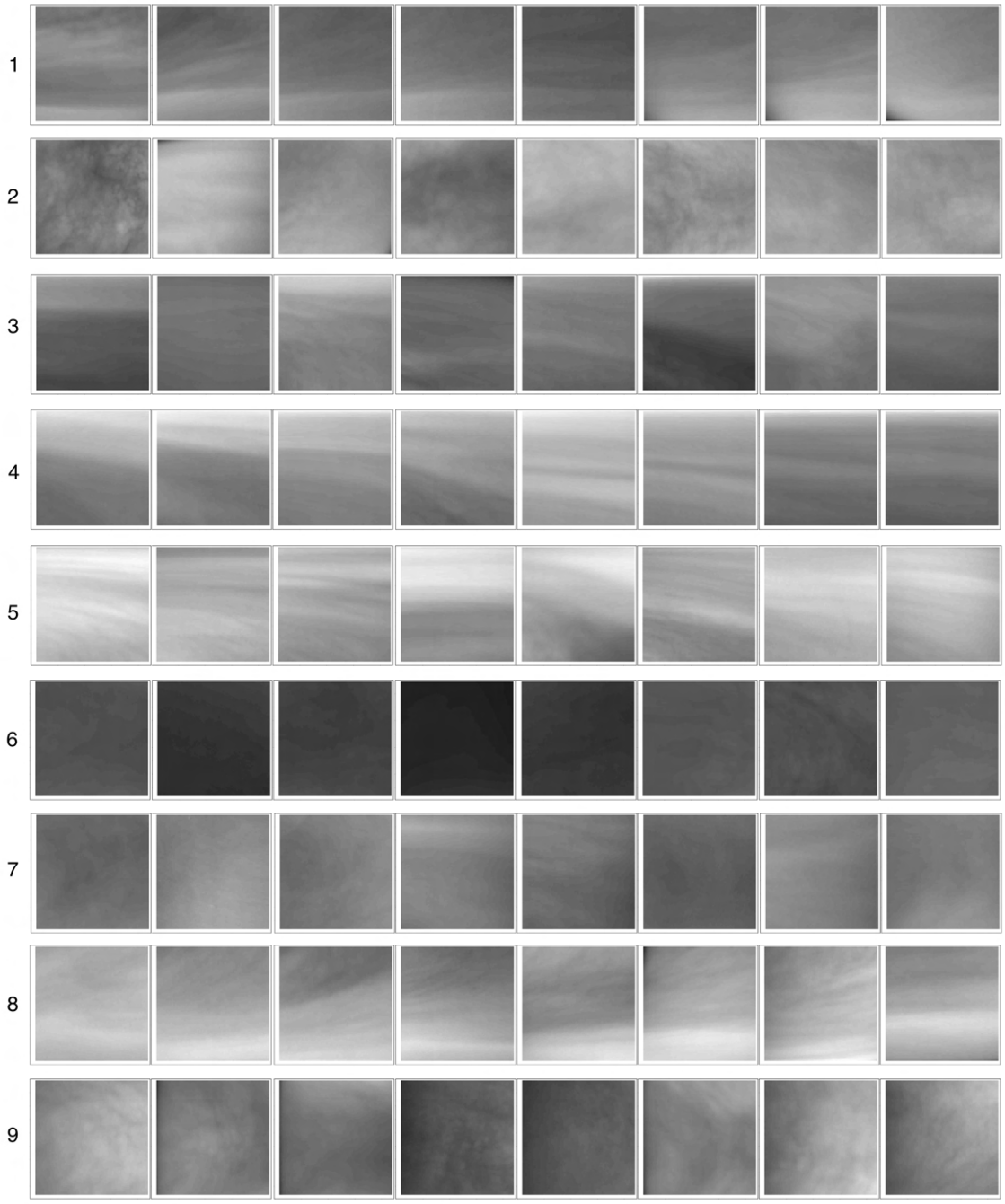

Figure 42: Results of the VGG16 on the Venus dataset, once it has be pre-processed. The 9 clusters have been shown. The algorithm learns how to distinguish the main features of the pictures, even though, due to the original dataset, not enough pictures are close enough to distinguish a variety of cloud morphology features.