Assignment 2: RL and Interpretability

Enrique Zaragoza Guajardo, student.

***Abstract*— In this document, a continuation of the work described in the assignment 1, was done. Previously, an Atari agent was used to obtain multiple observations of at least 6 different games. Using different methods, the interpretation of these observations was set as a goal. One tool, called LIME, was used; allowing to obtain a direct interpretation of the policy used by the agent during these observations. Then, optical flow provided a visual representation of the movement done during the different frames present in each observation. This also permitted to obtain a better understanding of the situation present that the agent faced. Finally, using algorithms form sci-kit learn, clusters were obtained which helped to create a relation between the observations and the interpretations given previously.**

***Index Terms*— Open-CV, Sci-Kit Learn, Cluster, Atari, TensorFlow, Tensorpack.**

I. Introduction

Previously, the role that Reinforcement Learning (RL) has been taking in the video games, was explained [1]. At the same time, explaining why it is important to give an interpretation of the different actions done by an agent; which serves as a purpose for the future experiments

.

It was stablished in [2] that most of the time the lack of interpretability in an agent's actions may not allow its improvement, or even tell if the structure of the actions is the done correctly.

For this reason, in the previous assignment, an algorithm present in the Tensorpack [3] library was used. This algorithm allowed to train, or run pretrained, agents with classic Atari games. For every situation an action was made by the agent. These actions were saved as observations, which contained 4 frames each. For these observations, 6 games were used: Alien, Asteroids, Breakout, Pong, Qbert and Seaquest; all which were present in the same library.

These observations served the purpose of obtaining a graphical representation of what a RL algorithm is capable to do when applied to videogames. The pretrained agents presented a decent performance in the videogames tested; nevertheless, as said in [2], to be able to identify the behaviour of an agent, interpretability [4] of its actions takes an important role in it. Different tools can be used to achieve this, being one of these LIME [5].

LIME (Local Interpretable Model-Agnostic Explanations) allows give a general interpretation of the reason behind an RL algorithm [5]. In this case, it will help to explain and understand the reasons behind every action of the agent in the Atari games. Using the observations, and the policy inside the agent, LIME is capable of comparing the situations and the actions made, giving gave an explanation to them; allowing to improve its interpretability.

Besides, because each observation is composed of 4 continuous frames, a visual representation of the movement between them could be useful to interpret them better. Thus, Optical Flow [6] can be used. Optical flow, being a tool from Open CV, allows to create a visual representation of the movement of different object between different frames of a video. In this case, the 4 different images inside each observation serves as the video frame. Using this type of visual representations, allows the user to obtain a better understanding of each observation; and comparing it to the interpretations obtained with LIME, it could help in its understanding.

Finally, using tools from sci-kit learn, clusters from the images could be obtained. This clusters could give a different approach to its interpretation. According to the number of clusters, and the data they have, a better perception of each situation could be obtained, thus, understanding the environment which the agent is facing.

II. Background

Various works have been done before, many which were mentioned in the last assignment, nevertheless some of them served as a guide and comparation for some experiments in this assignment.

To be able to understand the policy inside the agent, works such as the one in [7] were used. It explains how important is to create different models for different applications. In this case, it involves in how the interpretability different models can be an important role inside the improvement of these. This basic idea serves as inspiration of this assignment.

This document was received on Thursday 26 April 2018; as part of the assignment 2 of the module CE888: Data Science and Decision Making.

The model has the objective of the student to study different aspects of modern data science, giving it a use in different applications providing a complete study of it.

Dr. Spyros Samothrakis, Lecturer (R) and Assistant Director of IADS. School of Computer Science and Electronic Engineering (CSEE). Parkside block c2, Colchester Campus Telephone: +44 (0) 1206 872683 (email: ssamot@essex.ac.uk)

Different methods have been used to use RL in videogames, many which have focus in creating Atari Agents, such as [8] and [7], being the last one the most significant for this assignment. Using the work of [7] allowed to obtain the observations mentioned before.

The work in [5] explains the usage of LIME for different algorithms, and how different results can be obtained, such as visual and tabular interpretations. It allows to interpret the reasoning behind various machine learning algorithms, considering them as black boxes. Using their prediction policy, it helps to understand the reason behind most of the actions made.

For the visual representation of the movement in the frames, Optical flow [6] is a relevant tool. In the papers presented in [9] and [10], optical flow is used as a tool for movement detection, where in the first it is shown how it is used in a Raspberry Pi and, in the second, comparing it to other high-level computer vison algorithms. These documents, and specially the official tutorials in [6], helped to understand and use optical flow in the different experiments.

Finally, for clustering, tools present in sci-kit learn are the main objective. However, the work done in [11] and [12] give a working example of different uses to unsupervised learning algorithms, such as K-means and discretize spectral clustering with images.

III. Methodology

A. *LIME*

The official repository of LIME [13] offers different examples for the usage of this algorithm, from tabular to visual interpretation. In this case, the base for the code used was the one presented in the Image recognition example; which is an algorithm present in TensorFlow.

The example explains how the code must be initialized for the image recognition, and eventually how to obtain its policy so LIME is able to obtain the interpretation in how the algorithm is recognising the image.

In the case of the Atari agent two factors must be taken in consideration. First, the policy used in the Atari agent may be simple to understand, nevertheless, the model each game uses must be generated for the policy to work, for this reason some modifications to the original code must be done. Second, the observations previously obtained are in image format, png format to be specific; so, in order for the prediction to happen, a conversion of them has also to be done, so an acceptable input is achieved.

Having these two considerations, the algorithm in LIME should work as expected.

B. *Optical Flow*

Optical Flow offers different types of image visualization of the movement flow, the ones presented in [6] are Lucas-Kanade Optical Flow and Dense Optical Flow, each one offering different approaches for the output. For the purposes of this assignment, and the image quality of each observation, the Dense Optical Flow does not offer an ideal result of the movement flow; thus, the Lucas-Kanade Optical Flow method should be used.

For the code to work with the observations of the games, some modifications have also to be done. Using the examples in [6], [9] and [10], the camera input needs to be substituted with each observation, which is divided into 4 frames, then the flow from the first to the last frame should be obtained.

At the same time, using the functions in [14], the images could be modified to make them sharper, allowing the algorithm to have a more precise representation of the flow.

C. *Clustering*

Sci-kit learn offers different unsupervised learning algorithms, many which allow to obtain clusters. The examples presented in [11] and [12] use images preloaded inside the library of Sci-kit learn and, using k-means and discretisation techniques for clustering, inside Sci-kit learn as well, different segmentations or clusters can be obtained from the images.

In this case, each observation has to be transformed into the format that this type of clustering accepts, where cutting and separating the different frames could also help in the clustering.

IV. Experiments

A. *LIME*

To understand the use of this algorithm, first, one of the examples was used. The example, present in [5], shows how to use an image recognition algorithm and then LIME to interpret the reason behind each decision made.

The code is explained in different parts. First, the code needs to import all the different libraries for both LIME and the recogniser, but that is a standard thing to do in each code.

Then, the function of the recogniser is done with different functions, which include the policy of the predictor. This policy is important to LIME, so it can refer the results to the logic used by the predictor.

```
image_size = inception.inception_v3.default_image_size
def transform_img_fn(path_list):
…
```

```
checkpoints_dir = '/home/mlvm2/tf-models/slim/pretrained'
init_fn = slim.assign_from_checkpoint_fn(
    os.path.join(checkpoints_dir, 'inception_v3.ckpt'),
    slim.get_model_variables('InceptionV3'))
init_fn(session)

def predict_fn(images):
    return                          session.run(probabilities,
feed_dict={processed_images: images})
```

The last function called `predict_fn(images)` is basically what LIME will use to compare the results. It must be said, that in order for the code to work, a pretrained model needs to be downloaded, as it is shown previously. To obtain a more visual and complete idea of the code, it is recommended to see the Jupiter notebook attached in this assignment.

Using this code, and an image of a dog and a cat, the results obtained using the following code are shown in the Figure 1.

```
images = transform_img_fn(['cat_dog_portrait.jpg'])
plt.imshow(images[0] / 2 + 0.5)
preds = predict_fn(images)

for x in preds.argsort()[0][-5:]:
    print (x, names[x], preds[0,x])
```

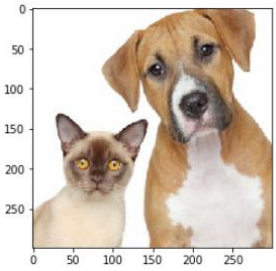

Figure 1. Results obtained with the image recognition algorithm. Giving the top prediction (with its possibility) as: Boxer (0.61), Bull Mastiff (0.04), American Staffordshire (0.02), Saint Bernard (0.01) and Rhodesian Ridgeback (0.01).

Then, once the algorithm has done its function, it is time for LIME to obtain an interpretation of it.

First, an object of LIME is created, and the interpretation is obtained.

```
explainer = lime_image.LimeImageExplainer()
explanation = explainer.explain_instance(image, predict_fn,
top_labels=5, hide_color=0, num_samples=1000)
```

It must be observed, that the explanation uses the explainer from LIME and the previous function for the prediction, called `predict_fn`, is used as well.

The result of this, allows to create labels, and in this case, give a visual explanation of the interpretation according to the parameters stablished.

Using the next code, an example of a label is shown, and the visual representation of it as well, shown in the Figure 2.

```
temp, mask = explanation.get_image_and_mask(X1,
positive_only=True, num_features=10, hide_rest=True)
plt.imshow(mark_boundaries(temp / 2 + 0.5, mask))
```

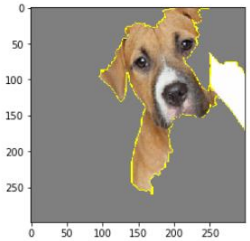

Figure 2. Explanation of the label “Boxer”, hiding everything but the prediction.

Once obtained this, it is time to apply the policy used with the Atari agent. For this reason, different steps have to be done.

The prediction function in the Tensorpack algorithm was easily shown in the `common.py` code, previously modified in the assignment 1. The function is called `predict(s)`.

```
def predict(s):
        """
        Map from observation to action, with 0.001 greedy.
        """
        act = pred(s[None, :, :,:])[0][0].argmax()
        return act
```

Nevertheless, this function needs an object called `pred`, which is a profile made for each individual game, that helps to predict the action that the agent needs to do according to the current situation.

For this reason, `train_atari` was modified, and according to the game, a profile was generated. To do this, `train_atari` is imported, the name of the game is defined, and the profile is created. For a more explicit demonstration, the Jupiter notebook of this code, and the code for `train_atari`, are annexed In the GitHub repository.

```
game ="Asteroids-v0"
from train_atari import *
pred = main2(game)
```

Now that `pred` is obtained, another step has to be done. As said before, the observations obtained are in image format, which is not an input accepted by the function `predict(s)`, which means that they need to be transformed.

The object `pred` uses an array of size [X,83,83,12] to obtain an action value for the situation presented. If the observation is divided in frames, each frame has an array value of [83,83,3], which means that if the for are stacked in a certain way, an array of [83,83,12] size could be obtained. For the first value, represented by an X, different values it can obtain are going to be explained further.

Thus, the function `grow(s,sizeFirst)` is created, which uses the observation, in array format, as an input, and returns an array of size [X,83,83,12].

Now, in order to be able to use the policy in LIME, the function `predict2()` needs to give a useful output to LIME. At this moment, this function returns an integer, value which is obtained by using `pred` and the observation, nevertheless, LIME needs an array of different values according to the

number of samples that are going to be used. For this reason, the variable `numberTimes` is used, which allows to give the X value in the function of `grow(s,sizeFirst)`, previously mentioned, and at the same time, allows the function `predict(s)` to loop and output the array needed for LIME.

Finally, the function `predict(s)` is modified as shown, and LIME is used in a similar way as before, using `predict2(s)` as the policy of the agent.

```
def predict2(s):
        """
        Map from observation to action, with 0.001 greedy.
        """
        sizeFirst, a,b,c = s.shape
        new_act   =   np.zeros((numberTimes,   numberTimes),
dtype="float32")
        s = grow(s, sizeFirst)
        i = 0
        while i < numberTimes:
            act = pred(s[None, :, :,:])[0][0].argmax()
            new_act[0][i]=act
            i +=1
        print (new_act.shape)
        return new_act

explainer = lime_image.LimeImageExplainer()
explanation = explainer.explain_instance(arr,
predict2,num_samples=numberTimes)
```

Using the different observations, different explanations were obtained, as shown in the figure 3. To show the interpretation of LIME, only the positive prediction is shown, and the background is not hidden; where the number of features of each varies.

B. Optical Flow

To create the optical flow of the different observations, the codes and examples presented in [6]; [9] and [10] were used as a guide to obtain what was needed for the assignment.
Each game, had a different configuration in the parameter for ShiTomasi corner detection, many which allowed to obtain better and clearer results. The parameters modified are the ones shown in the table 1.

Table 1. Parameters used for ShiTomasi corner detection in optical flow

Game	Max Corners	Quality Level	Min Distance	Block size
Alien	100	0.3	8	8
Pong	100	0.5	6	6
Asteroids	100	0.5	6	6
Breakout	100	0.2	2	2
Qbert	100	0.8	3	4
Seaquest	100	0.4	6	6

Once stablished the parameters for the game that was going to be used, a loop was created, allowing to use different observations to obtain different examples. Nevertheless, if one

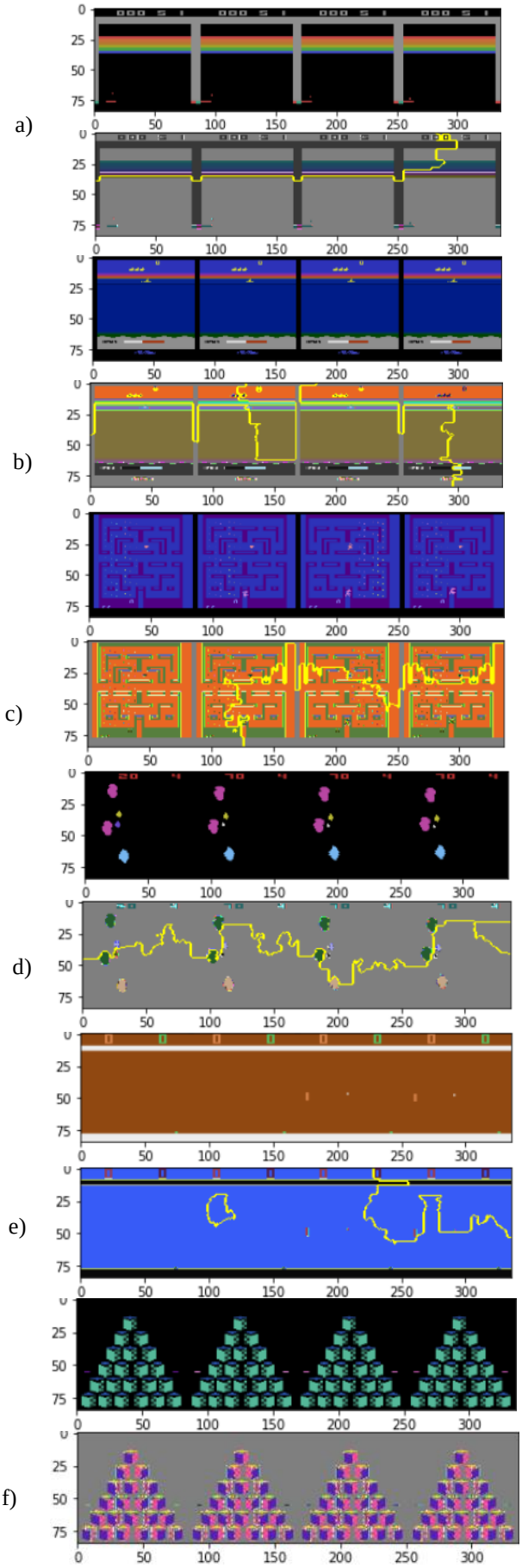

Figure 3. Interpretations obtained by LIME, where: a) Breakout with 10 features, b) Seaquest with 10 features, c) Alien with 100 features, d) Asteroids with 10 features, e) Pong with 5 features and f) Qbert with 5 features. For a bigger visualization of each image, see the document attached containing all images and screenshots from the code, as well as the Jupiter notebook called LIME_atari.

observation is needed to be explained, it should be the only observation to be analysed by LIME and optical flow.

Once read the observation, the different frames need to be separated. And, in order to obtain better results, a filter to these frames was also applied, to obtain sharper images. To do so, the general instructions are the following.

```
# cut and sharp the frame #
bbox = (0, top, division, height)
imageSlice = image_test.crop(bbox)
imageSlice = imageSlice.filter(ImageFilter.SHARPEN)
imageSlice.save("Slice#.png")
```

Once the four frames are obtained, the optical flow can be done. The first image is read, converted to grayscale and a filter with the previous parameters, described in table 1, is applied.

```
old_frame = cv2.imread(name)
old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
p0 = cv2.goodFeaturesToTrack(old_gray, mask = None,
**feature_params)
mask = np.zeros_like(old_frame)
```

After that, the next procedure is applied to the other 3 frames, doing the next steps, obtaining a final image showing the movement flow in the observation

First, the image is read and converted to grayscale.

```
frame = cv2.imread(name)
frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

The optical flow is then calculated, and the good points are selected

```
p1,    st,    err    =    cv2.calcOpticalFlowPyrLK(old_gray,
frame_gray, p0, None, **lk_params)
good_new = p1
good_old = p0
```

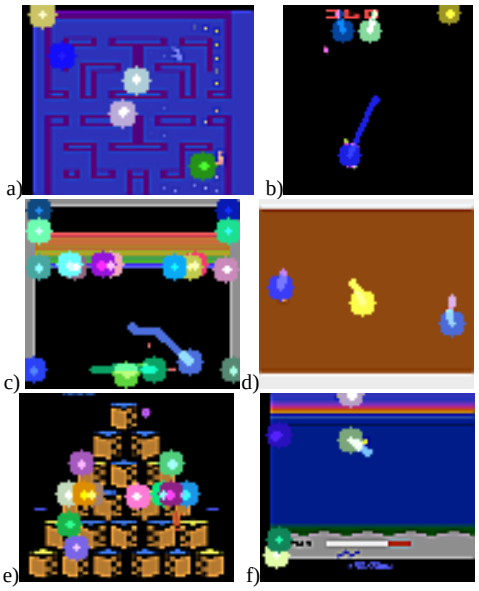

Figure 4. Representation of Optical Flow in observations from different games, where a) Alien, b) Asteroids, c) Breakout, d) Pong, e) Qbert and f) Seaquest. To see more examples of each games, see the attached folder in the GitHub repository.

Then, the tracks are drawn.

```
for i,(new,old) in enumerate(zip(good_new,good_old)):
   a,b = new.ravel()
   c,d = old.ravel()
   mask = cv2.line(mask, (a,b),(c,d), color[i].tolist(), 2)
   frame = cv2.circle(frame,(a,b),5,color[i].tolist(),-1)
img = cv2.add(frame,mask)
```

Then, the new image is saved

```
cv2.imwrite(file_path + "OF_"+game+str(counter)+".png",
img)
```

And finally, the previous point and base image are updated

```
old_gray = frame_gray.copy()
p0 = good_new.reshape(-1,1,2)
numberFrame = numberFrame + 1
```

Some of the results are shown in the Figure 4, where the most relevant and clearer examples were chosen.

C. Clustering

Finally, using the examples in [11] and [12], the clusters using the Sci-kit learn library are obtained.

First, the image is read, and converted to grayscale.

```
face = cv2.imread(name+".png")
face = cv2.cvtColor(face, cv2.COLOR_BGR2GRAY)
```

And the image is resized, so the unsupervised learning algorithm performs faster.

```
face = sp.misc.imresize(face, 0.99) / 255.
```

It is reduced to its 99% size, so it does not lose too much quality and the same time, performs faster.

Then, the graph of the image is obtained, which represent the gradient of the edges.

```
graph = image.img_to_graph(face)
```

Then, parameters are created to obtain the data form the graph. Where the beta value indicates how independent the segmentation is, and the exponential function makes the function of the gradient to decrease.

```
beta = 5
eps = 1e-6
graph.data = np.exp(-beta * graph.data / graph.data.std())
+ eps
```

Finally, after stablishing with a variable the number of clusters that are wanted, the clustering is done.

The labels are created using spectral clustering, where `N_REGIONS` is the number of clusters, or regions, and `assign_labels` indicates if whether `kmeans` or `discretize` will be used.

```
labels = spectral_clustering(graph, n_clusters=N_REGIONS,
as-sign_labels=assign_labels, random_state=1)
```

Then, the labels are applied to the image, and it is shown.

```
labels = labels.reshape(face.shape)
plt.figure(figsize=(5, 5))
plt.imshow(face, cmap=plt.cm.gray)
```

This is done for `kmeans`, and for `discretize` and each result is shown.

Some examples of this results, with the number of regions are shown in figure 5.

V. Discussion

It has been told that the main focus of this assignment is to provide applications to different tools to provide, or improve, the interpretability of the actions performed by an agent of an RL algorithm. In this case it was an agent capable of playing Atari games. For this reason, the experiments previously done obtained some important information about it.

A. *LIME*

As shown in the figure 2, LIME proves to be a good tool to provide explanations behind the decision of an image recogniser. It gives the opportunity to explain the reasoning behind many labels predicted, showing the positive and negative prediction, modify the number of features taken in consideration and isolate the predicted areas.

In the case of the Atari agent, the decisions made are represented by numbers, which are obtained by an argmax function. This means that the visual representation of each action is not as easy to achieve, which could be seen in the figure 3.

The different interpretations obtained may not be clear in some cases but may give a general idea of what is going on. For example, in the case of Qbert, it can be seen that the agent is focusing on the pyramid and nothing else, which means that the action generated is considering the circumstances occurring in the pyramid. In the other hand, in the cases of Alien and Seaquest, it is not very clear. A big part of the environment is highlighted, meaning that the agent is considering the whole environment and not a certain object; giving an interpretation that may not be useful. However, cases such as Breakout or Asteroids, the interpretation obtained shows that the action is focused in a certain area such as the line of bricks, or the incoming asteroids.

This may be easily improved by ensuring the correct input of observations into the predictor, because even though the code seems to work, there is not warranty that the actions predicted are done as expected; making it harder to interpret the actions and improve the pretrained agent.

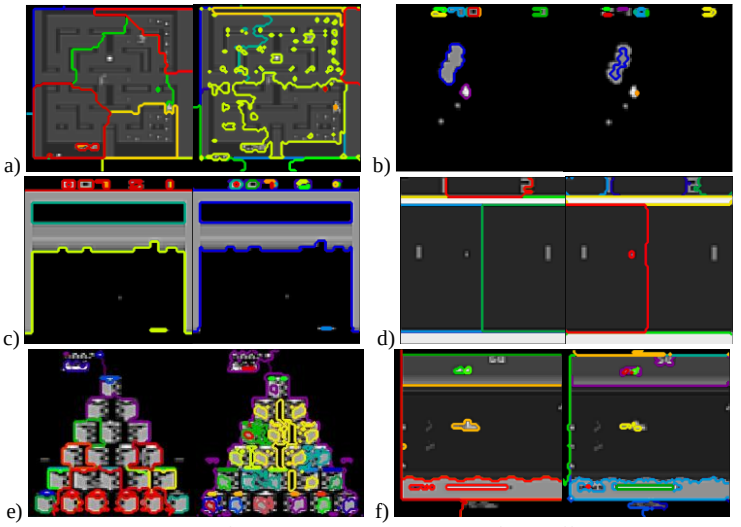

Figure 4. Representation of Optical Flow in observations from different games, where a) Alien 15 regions, b) Asteroids 10 regions, c) Breakout 15 regions, d) Pong 7 regions, e) Qbert 30 regions and f) Seaquest 13 regions. To see more examples of each games, see the attached folder in the GitHub repository.

To solve this problem, the predictor function needs to be confirmed to work as expected, as if it was when the observations were taken; and that the inputs generated by LIME are correct ones.

B. *Optical Flow*

The software provided by [6] proves to be a really useful tool to show a visual representation of the movement in a certain number of frames, with experiments shown in [9] and [10]. However, the observations provided have the problem to have low image quality. This means that some modifications to the images and tuning to the algorithm would be needed.

The first modifications made, is the appliance of a sharpening filter to the images, which proved to work successfully in the observations. However, the problem with the small size of the images was a problem for tuning the code; giving some outputs clearer than others.

Such is the case, in the figure 4, where the games of Breakout, Asteroids and Pong give a clear representation of how the agent and the environment is changing, leaving a clear trace of their movement. However, cases such as Alien and Qbert do not provide a so clear trace of movement. The images require more previous knowledge to understand the situation and obtain an interpretation from them. For example, in Alien, the agent, and the enemies of the game, are always highlighted, giving the user an easier understanding of where are they. And, in Qbert, it can be seen that the boxes, or steps, highlighted are the ones the agent is aiming to touch to complete the game. In the other hand, the case of Seaquest provides a mixture of both clear and unclear representation. Where it highlights the agent, and its enemies, but at the same time, provides a motion trace of some of them.

The previous problems could be solved by tuning better the algorithm, modifying the values presented in the table 1, taking care in not making it too sensible to the change in pixels between each frame.

C. Clustering

The examples provided in [11] and [12] give a good idea of what the different unsupervised learning algorithms present in sci-kit learn are capable of.

In the same way as it was done with optical flow, the sharpening of the images provided clearer results in obtaining clusters. The usage of two different types of spectral clustering, with K-means and discretise, allows to give two different perspectives of the observations.

Depending of the technique, and the number of segments, the different clusters allows to give different interpretations to the scenes. Having some clusters pointing to specific objects, where others point to areas. One example, present in the figure 5, is the one with Asteroids, where both methods provide selection of single elements, creating a possible interpretation that the agent could be focusing only in these. In the other hand, cases like Pong, Breakout and Seaquest, focuses more in areas and the agent itself; providing a possible opposite interpretation than before, where the agent could be focusing in these areas instead of certain objects. Finally, the case of Alien, one method provides a concentration in areas while the other method does this with objects, giving the user the option to choose the configuration that helps to give a better interpretation.

These three points of view may generate a more complete interpretation of the observations and the actions made by the agent, where LIME provides the reason behind every action in an observation, optical flow a visual representation of the movement in it and the clusters may point to relevant areas of the scenarios of the same. Meaning that each three may complement each other to obtain a more complete interpretation of an observation.

VI. Conclusion

During this project, different approaches have been taken to give a better idea about the importance with RL algorithms with videogames, and the interpretability behind their policies. Using a pretrained agent, different games of Atari were tested, and some observations were obtained. Each of these observations consisted in 4 frames of continuous moments in the game; which were useful for different experiments. Using tools such as LIME, Optical Flow and clustering techniques, interpretations of the actions and observations were obtained.

With the different results obtained, the importance of a clear interpretation behind an algorithm policy was demonstrated, showing if the predictions are not clear, it is uncertain to know if an agent is acting correctly, thus, making improvements hard to achieve. Furthermore, it was also shown why the interpretation of a policy should not be leaved only to a single tool, because different approaches to a same scenario could give a more complete perfective of it, providing more feedback for improvements.

References

[1] B. J. Min and K. J. Kim, "Learning to play visual doom using model-free episodic control," 2017 IEEE Conference on Computational Intelligence and Games (CIG), New York, NY, 2017, pp. 223-225.

[2] P. Lisboa. “Interpretability in Machine Learning – Principles and Practice”. Fuzzy Logic and Applications, vol. 8256, 2013.

[3] "Tensorpack", GitHub, 2018. [Online]. Available: https://github.com/ppwwyyxx/tensorpack. [Accessed: 11- Apr- 2018].

[4] Z. Lipton. (2016). “The Mythos of Model Interpretability.” ICML, Workshop on Human Interpretability in Machine Learning. New York, NY, USA. [Apr. 11, 2018].

[5] M. Tulio Ribeiro, S. Singh and C. Guestrin. (2016, Feb.). “"Why Should I Trust You?": Explaining the Predictions of Any Classifier”. ARXIV [Online]. eprint arXiv:1602.04938. Available: https://arxiv.org/pdf/1602.04938v3.pdf [Apr. 11, 2018].

[6] “Optical Flow.” Internet: https://docs.opencv.org/3.3.1/d7/d8b/tutorial_py_lucas_kanade.html, Oct. 24, 2017 [Apr. 11, 2018].

[7] S. Greydanus, A. Koul, J. Dodge and A. Fern. (2017, Oct.). “Visualizing and Understanding Atari Agents”. ARXIV [Online]. eprint arXiv:1711.00138. Available: https://arxiv.org/pdf/1711.00138.pdf. [Apr. 11, 2018].

[8] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg and K. Kavukcuoglu (2018, Feb.). “IMPA-LA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Archi-tectures”. ARXIV [Online]. eprint arXiv:1802.01561. Available: https://arxiv.org/pdf/1802.01561.pdf. [Apr. 11, 2018].

[9] R. Baby and R. Ahamed, "Optical Flow Motion Detection on Raspberry Pi", 2014 Fourth International Conference on Advances in Computing and Communications, pp. 1-3, 2014.

[10] M. Marengoni and D. Stringhini, "High Level Computer Vision Using OpenCV", 2011 24th SIBGRAPI Conference on Graphics, Patterns, and Images Tutorials, pp. 1-3, 2011.

[11] "Segmenting the picture of a raccoon face in regions — scikit-learn 0.19.1 documentation", Scikit-learn.org, 2018. [Online]. Available: http://scikit-learn.org/stable/auto_examples/cluster/plot_face_segmentation.html. [Accessed: 13- Apr- 2018].

[12] "Segmenting the picture of Lena in regions — scikit-learn 0.15-git documentation", Scikit-learn.org, 2018. [Online]. Available: http://scikit-learn.org/0.15/auto_examples/cluster/plot_lena_segmentation.html. [Accessed: 13- Apr- 2018].

[13] "LIME", GitHub, 2018. [Online]. Available: https://github.com/marcotcr/lime. [Accessed: 10- Apr- 2018].

[14] "ImageFilter Module — Pillow (PIL Fork) 3.1.2 documentation", Pillow.readthedocs.io, 2018. [Online]. Available: http://pillow.readthedocs.io/en/3.1.x/reference/ImageFilter.html. [Accessed: 13- Apr- 2018].