MACHINE LEARNING CAPSTONE PROJECT

CAPSTONE Project - Machine Learning Engineer Nanodegree - October 2020

Project location : https://github.com/diegoami/DA_ML_Capstone

DEFINITION

PROJECT OVERVIEW

During the last few years it has become more and more common to stream on platforms such as Youtube and Twitch while playing video games, or to upload recorded sessions. The volume of videos produced is overwhelming. In many of the video games being streamed there are different types of scenes. Both for content producers and consumers it would be useful to be able to automatically split videos, to find out in what time intervals different types of scenes run. For instance, having as an input the video recording of a Minecraft speedrun, we could be able to produce the time intervals when the game is taking place in the Overworld surface, in caves, in the Nether and the End respectively - the four main settings of this game.

The game that I have chosen to analyze is *Mount of Blade: Warband*, of which I made several walkthroughs. This is a game where you spend most of the time on a "strategic map", taking your warband to any of the towns or villages, following or running away from other warbands which can belong to friendly or rival factions, or looking for quest objectives.

(a) Close to a town

(b) Bird's view

(c) Your warband

Other in-game screenshots can show the character's inventory, warband composition, allow interaction with non-playing characters (NPCs), display status messages...

(a) In a shop

(b) Manage your warband

(c) Talk to a NPC

The hero can also take a walk in town, villages and castles, have training sessions with soldiers, or spar with them in arena.

(a) In a noble's castle (b) Training Session (c) Challenge in the arena

However, what we are interested in is locating the scenes when the warband engages enemies and the game switches to a tactical view, such as a battle in an open field or in a village...

(a) Raid on caravan (b) Major battle (c) Desert battle

or when they sieges a town or a castle...

(a) Unuzdaq castle (b) Dhirim (c) Halmar

or when they assault a bandit hideout.


(a) Forest bandits


(b) Tundra bandits


(c) Steppe bandits

The hero often takes part in tournaments, which are a very important part of the game which we want to locate. Regrettably, they may look similar to situations like training or sparring in the arena, that are not very interesting.

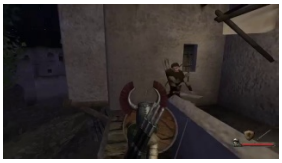

(a) Tulga tournament


(b) Tulga tournament


(c) Tulga tournament

Quests and ambushes, are pretty infrequent and the screenshots may look similar to those of more peaceful situations. For instance when the hero is rescuing a lord from prison, or fighting a bandit in a village, are not very different from scenes when he might be just taking a stroll in a town or village.


(a) Lord Rescue


(b) Ambush in Town


(c) Ambush in village

PROBLEM STATEMENT

The goal is to create and deploy a model which is able to classify images from the game *Mount&Blade: Warband* and return a category, such as "Battle", "Hideout", "Siege", "Tournament" and "Other". It is also desirable to find out an optimal number of categories. It would be ideal to have categories for less frequent scenes such as "Prison escape", "Ambush", "Quest", but this will be out of scope and such scenes will be lumped together with the closes "main" category.

An additional goal is to have a model which identifies contiguous scenes in a gameplay video of *Mount&Blade: Warband*, providing the beginning and the end of the scenes, and its category.

A necessary requirement for this project is to gather a dataset of screenshots taken from the game, as well as the category to which they belong.

METRICS

I will measure the performance of the image classifier using accuracy and cross-entropy loss on the training, validation and test (holdout) dataset.

I will also measure precision, recall, accuracy and F1 for each category, as well as a total weighted and mean accuracy. I will also provide a confusion matrix for each class.

ANALYSIS

DATA EXPLORATION

CREATING A DATASET To create a dataset I took some videos from a game walkthrough of mine, the adventures of Wendy. I used the episodes from 41 to 68 from following playlists on youtube:

- CNN-Wendy-I: *https://www.youtube.com/playlist?list=PLNP_nRm4k4jfVfQobYTRQAXV_uOzt8Bov*
- CNN-Wendy-II: *https://www.youtube.com/playlist?list=PLNP_nRm4k4jdEQ-OM31xNqeE64svvx-aT*
- CNN-Wendy-III: *https://www.youtube.com/playlist?list=PLNP_nRm4k4jdEQ-OM31xNqeE64svvx-aTPLNP_nRm4k4jeoJ8H7mtTUbbOJ6_Rx_god*

I found scenes in these episodes and added scene descriptions, that can be found in the video descriptions on youtube.

For instance, in episode 54, I have identified following scenes, of the category "Hideout", "Battle", "Tournament", "Town" ("Town" is eventually remapped to "Battle"). All the other parts of the video are categorized as "Other". These lines can be found in the video description.

- 09:51-12:21 Hideout Tundra Bandits (Failed)
- 18:47-19:44 Battle with Sea Raiders
- 20:50-21:46 Battle with Sea Raiders
- 22:54-23:42 Battle with Sea Raiders
- 34:06-37:44 Tournament won in Tihr
- 38:46-40:48 Town escape for Boyar Vlan

COMPANION PROJECTS To prepare the data set, I had set up a companion project under *https://github.com/diegoami/DA_split_youtube_frames_s3/*.

This project:

- downloads the relevant videos from youtube, using the *youtube-dl* python library, in a 640x360 format
- extracts at every two seconds a frame and save it an jpeg file, using the *opencv* python library, resizing to the practical format 320x180
- downloads the text from the youtube description and save it along the video (*metadata*)
- Copy files to directories named by the image categories.

The result of this process has been uploaded to Amazon S3. Files can be browsed here: https://da-youtube-ml.s3.eu-central-1.amazonaws.com/

DATASET CHARACTERISTICS The dataset contains 51216 images , 320 x 190, in jpeg format, categorized in this way

Category	Amount	Percentage
Battle	7198	14.0%
Hideout	1163	2.2%
Other	35425	67.4%
Siege	634	1.2%
Tournament	6796	13.3%
TOTAL	51216	

You can browse them using the *analysis.ipynb* notebooks. While having a look at images, it seemed to me that a format of 320x180, in grayscale, keeps most of the information necessary to categorize it.

EXPLORATORY VISUALIZATION

Using PCA on a flattened version of the image matrixes, in format 80x45, black and white, I produced visualizations in 2d of the dataset.

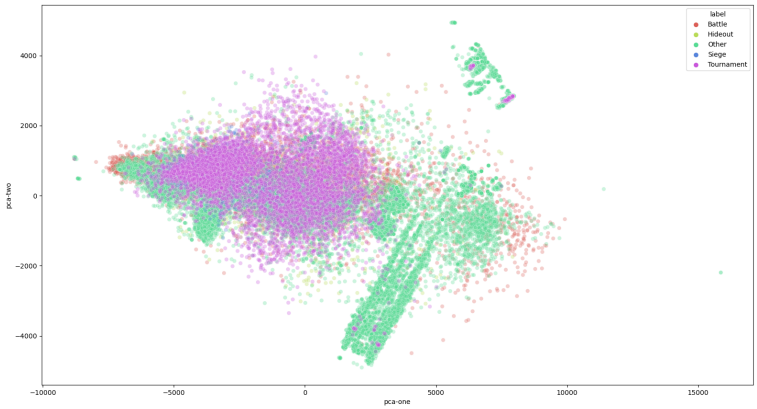
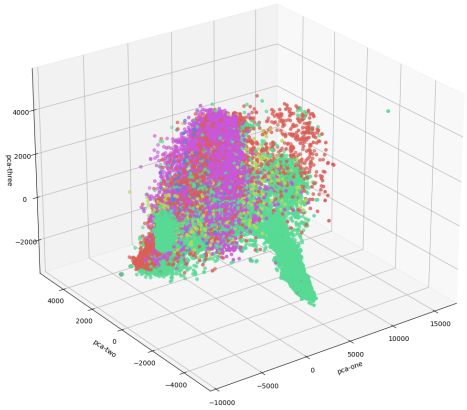


Figure 9: pca_sklearn_2d.png



and here in 3d, using the same color scheme.

It can be seen that “Other” scenes are separated in several clusters. “Battle”, “Tournament”, “Siege” and “Hideout” images do group in certain regions, but there is a lot of overlap beween each other and with some of the “Other” images.

We move on to create a VGG13 model and do a PCA representation of the features of the images dataset recovered from the last layer .

Here there are is much less overlap between regions where the different classes are located. That shows how the new created features are important for the categorization task.

- Other –> Tournament: possibly Arena and Training scenes (categorized

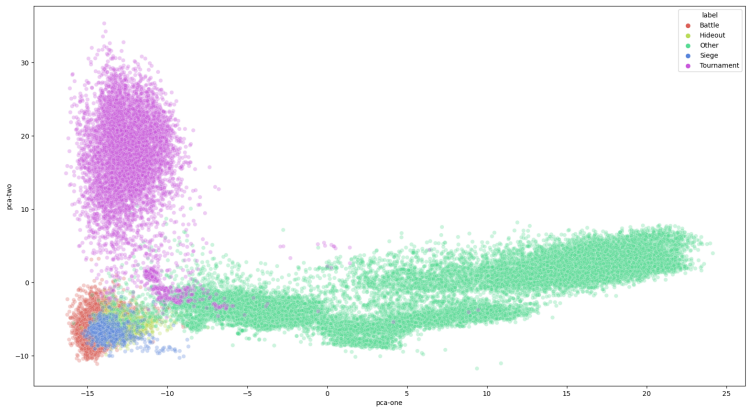


Figure 10: pca_v5d_2d_f.png

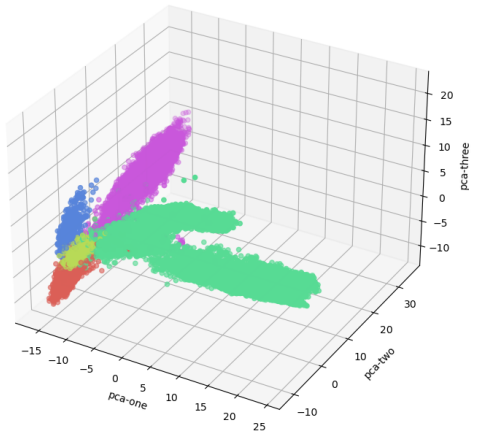


Figure 11: pca_v5n_3d_f4.png

as Other, they are similar to Tournaments)
- Battle –> Other : possibly Town escapes and Ambushes (categorized as Battle, but look like scenes in which the hero is strolling)

There is not much to do about that as we have too few images that we could categorize as Arena, Ambush, Training... and we will accept that those frames may confuse the model.

ALGORITHMS AND TECHNIQUES

GENERAL APPROACH The general idea is to create an image classifier to categorize the images extracted from gameplay videos as belonging to a particular type of scenes. Then to use this image classifier on frames extracted from other gameplay vidoes to identify how to split videos into scenese.

For the Image Classifier I choose to start with one of the Convolutional Neural Network which are already available in Pytorch, as

- this is a well-tried way to approach the problem
- I could use standard CNN topologies available in Pytorch, such as VGG, which would als not require too much memory
- analyzing the result of the model would give me more information on what I would have to be looking for

A simple to use and flexible topology I decided to use was VGG, which is powerful enough and small enough to fit on the GPU I used for training.

As the images extracted from game walkthroughs are not related to real world images, using a pre-trained net and expanding it with transfer learning did not seem a sensible approach. Instead, I opted for a full train.

MODELS The approach I considered most simple and promising was to use Convolutional Neural Network included in the torchvision package in Pytorch.

I ended up using the VGG13 Model, which gives satisfying results while also not being overbloated.

BENCHMARK

As 67.4 % of the images belong to the category "Other", a model should have an accuracy of at least 68% for being considered better than a model that always pick the Category "Other".

Other than that, I create a very simple model that would use flattened matrixes of images, possibly in greyscale, using standard scikit-learn transformers and PCA.

Using greyscale images in 80x45 format, I got the following results using a RandomTreeForest an SGD on 50 PCA-produced features, on the validation set (which had not been used for training).

It is not surprising that these results do not look that bad at all, and are actually good as separating "Other" images from images depicting any kind of engagement / fight (Tournament, Battle, Siege, Hideout). That is expected, as there are some GUI components that appear only in these scenes.

RandomForestClassifier

```
RandomForestClassifier(n_estimators=200)
Accuracy: 0.932
F1 Score: 0.926
```

	precision	recall	f1-score	support
0	0.83	0.91	0.87	2375
1	0.99	0.18	0.31	384
2	0.98	0.97	0.98	11691
3	1.00	0.34	0.51	209
4	0.81	0.92	0.86	2243
accuracy			0.93	16902
macro avg	0.92	0.67	0.70	16902
weighted avg	0.94	0.93	0.93	16902

Confusion matrix:

X	0	1	2	3	4
0	2154	0	79	0	142
1	106	70	26	0	182
2	165	0	11398	0	128
3	76	0	15	71	47
3	98	1	76	2	2068

SGDClassifier

```
SGDClassifier(alpha=0.001, max_iter=10000)
Accuracy: 0.841
F1 Score: 0.843
```

	precision	recall	f1-score	support
0	0.72	0.53	0.61	2375

1	0.53	0.37	0.44	384
2	0.97	0.93	0.95	11691
3	0.56	0.30	0.39	209
4	0.52	0.83	0.64	2243
accuracy			0.84	16902
macro avg	0.66	0.59	0.60	16902
weighted avg	0.86	0.84	0.84	16902

Confusion matrix:

X	0	1	2	3	4
0	1250	82	74	22	947
1	61	143	11	5	164
2	222	8	10884	19	558
3	67	26	0	62	54
3	135	13	225	2	1868

METHODOLOGY

DATA PROCESSING

The image dataset has been created extracting frames from video on youtube, and putting them in subdirectories named like the category they belong with. This is the main dataset I have been working with. These images have also been resized are in 320 x 180 RGB format , and normalized.

While training the benchmark models, images are transformed into grayscale and resized to 80 x 45, but when training the deep learning models we use the original images in color, 320x180.

As these images originate from video games, any kind of data augmentation such as mirrored or cropped images do not make sense. These images would not be produced by the game, as many GUI components are always in the same place.

IMPLEMENTATION

I set up scripts and notebooks so that they would work both locally and on Sagemaker.

A pytorch/conda environment, as the one in Sagemaker, is assumed - the missing libaries from the default sagemaker conda pytorch environment are in the */requirements.txt* file. The four environment variables require for Sagemaker, SM_HOSTS, SM_CHANNEL_TRAIN, SM_MODEL_DIR, SM_CURRENT_HOST, must be set.

The code root directory is *letsplay_classifier* - scripts should be executed from this directory, and directory should be included in PYTHONPATH.

For more details see the file README.md

TRAINING SCRIPT The training script *train.py* accepts following arguments:

- img-width: width to which resize images
- img-height: height to which resize images
- epochs: for how many epochs to train
- batch-size: size of the batch while training
- layer-cfg: what type of VGG net to use (A, B, C or D)

These are the steps that are executed:

- use an image loader from pytorch to create a generator scanning all files in the data directory.
- use a pytorchvision transformer to resize images
- divide the dataset in train, validation and test sets, using stratification and shuffling
- load a VGG neural network, modified so that the output layers produce a category from our domain (5 categories in the final version)
- For each epoch, execute a training step and evaluation step, trying to minimize the cross entropy loss in the validation set
- Save the model so that it can be further used by the following steps
- Evaluate the produced model on the test dataset, and print classification report and confusion matrix

The cross entropy is the most useful metrics while training a classifier with C classes, therefore it is used here.

VERIFICATION SCRIPT The verification script *verify_model.py* works only locally, as it assumes the model and the dataset is saved locally from the previous step. It requires the same environment variables as the training script.

- Loads the model created in the previous step
- Walks through all the images in the dataset, one by one, and retrieve the predicted label
- Print average accuracy, a classification report based on discrepancies, a confusion matrix, and a list of images whose predicted category does not coincide with their labels, so that the dataset can be corrected.

PREDICTOR The file *predict.py* contains the methods that are necessary to deploy the model to an endpoint. It works both locally and on a Sagemaker container and requires a previously trained model.

- input_fn: this is be the endpoint entry point, which converts a Numpy matrix payload to a Pillow Image, applying the same preprocessing steps as during training
- model_fn: predicts a category using a previously trained model, accepting an image from the domain space (a screenshot from *Mount&Blade: Warband* in format 320 x 180) and transformed in the same way as during training
- output_fn: returns the model output as a numpy array: a list of log probabilites for each class

ENDPOINT The file *endpoint.py* contains a method *evaluate* allowing to call an endpoint on Sagemaker, so that you can collect predictions, which can be used to show a classification report and a confusion matrix. It requires locally saved data, but the model is accessed through a published endpoint, unlike the *verify_model.py* component which requires a saved model locally.

endpoint.py works only in Sagemaker, when called from a Jupyter Notebook. Examples can be seen in the jupyter notebooks, for instance in CNN_Fourth_i teration.ipynb

JUPYTER NOTEBOOKS These are the jupyter notebooks I created while making this project:

- *analysis.ipynb*: just to browse through images
- *CNN_First_Iteration.ipynb* : First iteration with 8 classes
- *CNN_Second_Iteration.ipynb* : Second iteration with 5 classes and some corrections in the data set
- *CNN_Third_Iteration.ipynb* : Third iteration with more corrections, a more advanced model and a first attempt to use the model to split videos
- *CNN_Fourth_Iteration.ipynb* : Fourth iteration and final results

REFINEMENT

At the beginning I was trying recognize 8 types of scenes. Apart from the categories BATTLE, HIDEOUT, OTHER, SIEGE, TOURNAMENT I had also the categories TOWN, TRAP and TRAINING. But as can be seen in CNN_Fir st_Iteration.ipynb, I had too few amples of these and merged TRAINING into OTHER, while TOWN and TRAP became BATTLE.

Another important refinement was finding images in the dataset that had been wrongly labeled. For this, I used the output from one of my scripts, *verify_model.py*, which pointed to images whose prediction mismatched with labels. The only semnsible way to deal with that was to correct metadata at the source and use the companion project
DA_split_youtube_frames_S3 to regenerate frames in the correct directory.

I decided pretty early that my main model would be a VGG implementation in Pytorch.

RESULTS

MODEL EVALUATION AND VALIDATION

IMAGE CLASSIFICATION The final model uses 51216 images (320 x 180) in color, VGG13, with 8 epochs. This is the confusion matrix and metrics on the full dataset.

Confusion matrix

X	0	1	2	3	4
0	7126	13	49	3	7
1	0	1159	1	2	1
2	163	13	35154	5	90
3	5	6	0	623	0
3		181	181	2	6609

Classification Report

class name	class	precision	recall	f1-score	support
Battle	0	0.98	0.99	0.98	7198
Hideout	1	0.97	1.00	0.98	1163
Other	2	0.99	0.99	0.99	35425
Siege	3	0.98	0.98	0.98	194
Tournam	4	0.99	0.97	0.98	6796
	macro avg	0.98	0.99	0.98	51216
	weighted avg	0.99	0.99	0.99	51216

When just executed on the test dataset, it returns an accuracy of 98,7% and a cross entropy of 0.0026 and following confusion matrix and classification report

Confusion Matrix

X	0	1	2	3	4
0	995	3	7	1	2
1	0	163	0	0	0
2	26	3	4911	2	18
3	3	1	0	85	0
4	0	0	24	0	927

class name	class	precision	recall	f1-score	support
Battle	0	0.97	0.99	0.98	1008
Hideout	1	0.96	1.00	0.98	163
Other	2	0.99	0.99	0.99	4960
Siege	3	0.97	0.96	0.96	89
Tournam	4	0.98	0.97	0.98	951
	macro avg	0.97	0.98	0.98	7171
	weighted avg	0.99	0.99	0.99	7171

JUSTIFICATION

In the final model, both F1 and Accuracy are in the 97-99% range for every category, unlike the simple model I used in the benchmark. It is particularly important to be able to tell tournaments from battles, and show good precision / recall on Siege / Hideout frames, which are actually the information that we need to.

To see that, we can compare how the VGG13, Random Tree Forest and Stochastic Tree Descent scores compare in regard to classifying each category. We can have a look at the F1 score.

F1	VGG13	RTF	SGD
Battle	0.98	0.87	0.61
Hideout	0.98	0.31	0.44
Other	0.99	0.98	0.95
Siege	0.96	0.51	0.39
Tournament	0.98	0.86	0.64

The difference in Recall is even greater:

Recall	VGG13	RTF	SGD
Battle	0.99	0.91	0.53
Hideout	1.00	0.18	0.37
Other	0.99	0.97	0.93
Siege	0.96	0.34	0.30
Tournament	0.97	0.92	0.83

In future improvements I am planning to extract more categories, which will also have few samples. This makes the difference in Recall even more critical.

RESULTS

In the end, I opted for a VGG13 model (layer configuration "B" in pytorch) trained on full image size (320 x 180). I used for that a corrected version of the dataset (*https://da-youtube-ml.s3.eu-central-1.amazonaws.com/wendy-cnn/frames/wendy_cnn_frames_data_2b.zip*) with fewer misclassified images. The results refer to runs I executed locally on my computer. On Sagemaker the results are somewhat different, as can be seen in the *CNN_Third_iteration.ipynb* notebook, but I could not figure out why.

IMAGE CLASSIFICATION

When training for 5 epochs, this approach gives a 98.5 % accuracy and a cross entropy loss of 0.0035 both on train and validation set. The improvements are due both to cleaning the dataset and using a bigger format for images.

Classification report on full dataset: Accuracy = 0.99

class name	class	precision	recall	f1-score	support
Battle	0	0.97	0.99	0.98	6125
Hideout	1	0.99	0.97	0.98	1162
Other	2	0.99	0.99	0.99	31430
Siege	3	0.85	0.99	0.92	195
Tournam	4	0.99	0.97	0.98	6798
	macro avg	0.96	0.98	0.97	45710
	weighted avg	0.99	0.99	0.99	45710

Confusion Matrix

X	0	1	2	3	4
0	6055	0	55	8	7
1	4	1131	5	22	0
2	153	13	31212	2	50
3	0	0	1	194	0
4	8	2	197	1	6590

INTERVAL IDENTIFICATION

However, this is not the only result I was striving for. I wanted to create a tool not just to categorize images, but to split videos in scenes. Now, this problem would be worth a project in itself, possibly building a model on top of another model, or maybe considering RNN. At the moment I think this would make

the problem too complex, as I expect this tool just to be able to help redact description, and not create them without human supervision.

To convert image classifications to scenes, I use so far an empirical procedure which is very dependent on the model and the dataset. Using probabilities calculated on each frame, I set up a "string visualization" for each frame, such as `HHHSSSSSSBBBBBBBBBB`, which in this case would mean "50% battle, 15% Hideout, 35% Siege", or "_____TT" for " 90% Other, 10% Tournament". Scrolling these visualizations it is possible to get an overview of the scenes in the youtube videos.

The algorithms are in the *interval* package: locally I use (*predict_intervals_walkdir*), on Sagemaker you can use: *predict_intervals_endpoint*).

I applied this procedure to a few episodes whose images hadn't been used for training, from E69 to E77, skipping E70 which is an outlier (Siege Defence in Slow motion, bugged by the mod I was working on). The results, along with the correct metadata, can be found in the split_scenes directory of the main repository.

Pretty much all battles, sieges, hideout and tournaments are recognized correctly, apart form a couple of cases where there was some noise, explained later. For instance

```
...
41:10  __BBBBBBBBBBBBBB___
41:12  _BBBBBBBBBBBBBBBBBB
41:14  _BBBBBBBBBBBBBBBBBS
41:16  __BBBBBBBBBBBBBBBBB
41:18  _BBBBBBBBBBBBBBBBBB
41:20  __BBBBBBBBBBBBBBBBB
41:22  __BBBBBBBBBBBBBBBHS
41:24  _BBBBBBBBBBBBBBBBBB
41:26  _BBBBBBBBBBBBBBBBBB
41:28  __BBBBBBBBBBBBBBB_T
41:30  _BBBBBBBBBBBBBBBB_T
41:32  __BBBBBBBBBBBBBB__T
41:34  ___BBBBBBBBBBBBBBHH
41:36  __BBBBBBBBBBBBBB__T
41:38  ___BBBBBBBBBBBBBBBH

01:44-03:06 | Battle : 95%
19:50-21:54 | Battle : 89% , Other : 7%
36:50-38:04 | Battle : 94%
41:10-41:38 | Battle : 85% , Other : 10%
```

There are some instance where the output is less than perfect, but very helpful nonetheless. For instance, in E73 the hero talks a walk around her castle between 8:18 and 8:54, with no enenies, which usually would be an "Other" scene, but it

is very similar to some outdoor scenes. Ideally there should be a "Walk" category for that. Same issue in E74 and E75.

```
08:22  __HHHHHHHHHHHHHHHHHS
08:24  __HHHHH_____SSSSSTTT
08:26  __HHHH__SSSSSSSSSSS

08:30  __HHHH________SSSSSS
08:32  __HHHHH_SSSSSSSSSST

08:44  ____________SSSSSSS

08:48  __________SSSSSSSSS
08:50  _____SSSSSSSSSSSSS

08:22-08:26 | Hideout : 22% , Other : 27% , Siege : 42% , Tournament : 7%
08:30-08:32 | Hideout : 25% , Other : 15% , Siege : 55%
08:44-08:44 | ?????
08:48-08:50 | Other : 25% , Siege : 75%
```

Other interesting problems:

- in E73, a siege is recognized correctly, but the program is confused when the hero gets knocked out and the troops keep fighting without her, so the correct interval is not recognized.
- In E72 there are a couple of mountain battles, when the hero can fight dismounted, which are therefore then partly classified as Hideouts
- In E72 some noise is generated because of a "Training" session, which I usually put in the "Other" bucket, but are actually fight scenes with some similarity to Tournaments. Training would also deserve its own category, if there were more samples.
- In E73 the algorith gets confused during a siege when the hero gets knocked out
- In E72 and E74 the algorithms gets confused by short scene where the main

- The Siege of Halmar in E75 gets about 45% probability Siege, 40% probability Hideout
- In E76 some sparring in the Arena is confused with a Tournament

The idea in the long term is to gather more data and see what categories can also be introduced.