Automatic splitting into scenes of gameplay videos

https://github.com/diegoami/DA_ML_Capstone

Diego Amicabile

Final Capstone Project for the Machine Learning Engineering Nanodegree

Udacityb

2 November 2020

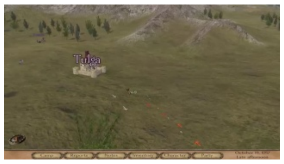
DEFINITION

PROJECT OVERVIEW

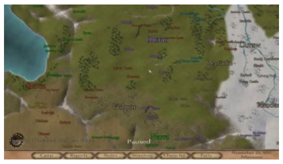
During the last few years it has become more and more common to stream on platforms such as Youtube and Twitch while playing video games, or to upload recorded sessions. The volume of videos produced is overwhelming. In many of the video games being streamed there are different types of scenes. Both 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.

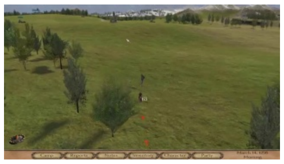
The game that I have chosen to analyze is *Mount of Blade: Warband*, of which I attempted several walkthroughs. In this game, you command a hero, and later on a warband, in their quest to power in the fictitious continent of Calradia, divided between six warring factions. On their way there you have to fight battles, sieges, tournaments, clear bandit handouts, fulfill quests in what is usually a very long game, spanning hundreds of hours.

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.

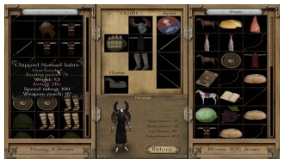


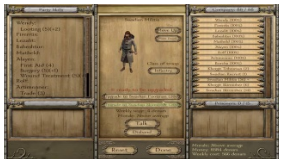
(a) Close to a town

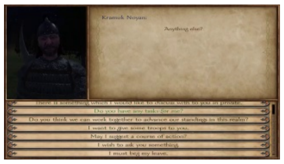

(b) Bird's view


(c) Your warband

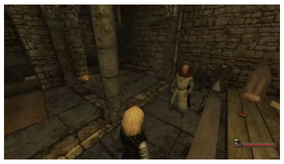
Other in-game screenshots can show the character's inventory, warband composition, allow interaction with non-playing characters (NPCs), display status messages. . .

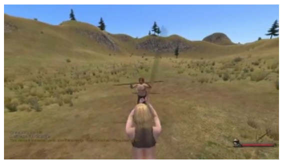

(a) In a shop

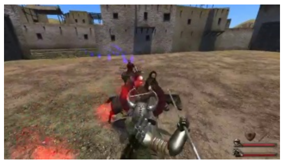

(b) Manage your warband


(c) Talk to a NPC

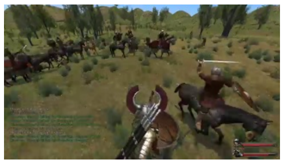
The hero can also take a walk in town, villages and castles, have training sessions with soldiers, or spar with them in arena.

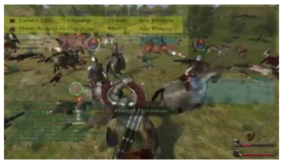

(a) In a noble's castle

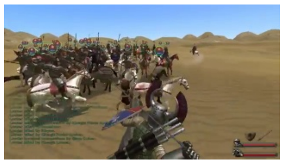

(b) Training Session


(c) Challenge in the arena

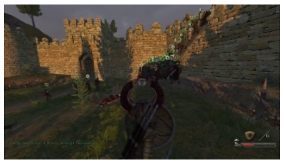
However, what we are interested in is locating the scenes when the warband engages enemies and the game switches to a tactical view, such as a battle in an open field or in a village…

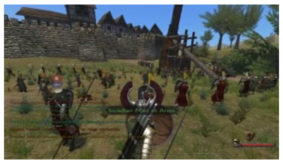

(a) Raid on caravan

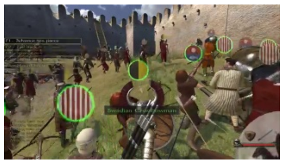

(b) Major battle


(c) Desert battle

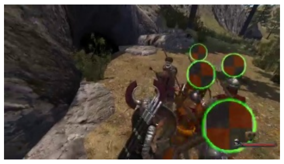
or when they siege a town or a castle...

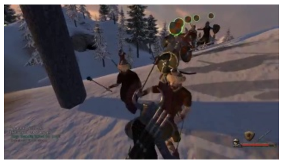

(a) Unuzdaq castle

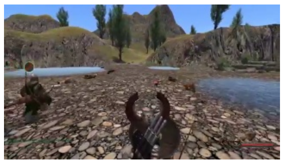

(b) Dhirim


(c) Halmar

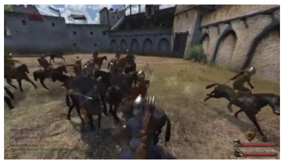
or when they assault a bandit hideout.

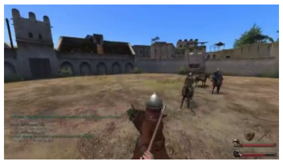

(a) Forest bandits

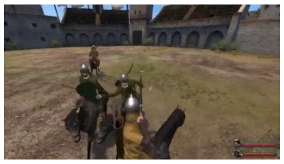

(b) Tundra bandits


(c) Steppe bandits

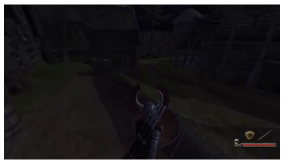
The hero often takes part in tournaments, which are a very important part of the gam, and which we want to locate in gameplay videos. Regrettably, they may look similar to situations like training or sparring in the arena, that are not very interesting.

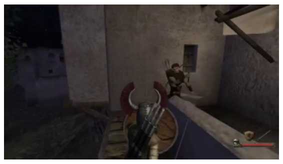

(a) Tulga tournament

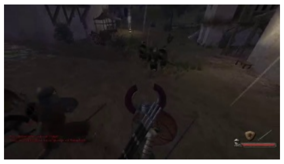

(b) Tulga tournament


(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” for each frame.

The approach should be robust enough to allow for later expanding of categories, for which there are not enough samples at the moment. 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 closest 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 their most likely categories.

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

METRICS

While training, validating and testing the image classifier model, I will evaluate it by means of accuracy and cross-entropy loss. While overall accuracy is not a good metrics in an unbalanced dataset, cross-entropy loss is a good loss function for Classification Problems with several classes, as it minimizes the distance between the predicted and the actual probability.

I will also use scikit-learn functions to measure precision, recall, accuracy and F1, for each class and on average, as well as to calculate a total weighted and mean accuracy. A model will not be considered good enough if it cannot deliver a good precision and, even more importantly, recall on classes that have relatively few samples.

ANALYSIS

Most files referenced in these section are relative to this Github repository..

DATA EXPLORATION

CREATING A DATASET

To create a dataset I took some videos from a game walkthrough of mine, the adventures of Wendy (I, II). I used the episodes from 41 to 68 to build the model, keeping episodes 69 to 77 to test it, and to simplify download I also created following playlists on youtube, containing just the episodes I will be using in this project:

- CNN-Wendy-I
- CNN-Wendy-II
- CNN-Wendy-III

To build a dataset, I identified scenes in these episodes and wrote them all down in the video description on youtube.

For instance, in episode 54, I have wrote down following scenes in the description, of the category "Hideout", "Battle", "Tournament", "Town" ("Town" is eventually remapped to "Battle") - first word of the description line. All the other parts of the video are categorized as "Other".

- 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 PROJECT

To prepare the data set, I created a companion project allowing to:

- download the relevant videos from youtube, using the *youtube-dl* python library, in a 640x360 format
- extract at every two seconds a frame and save it in *jpeg* file, using the *opencv* python library, resizing to the practical format 320x180
- download the text from the youtube descriptions, which describes the scenes, and save it along the video as *metadata*
- assign the images created in the second step to labels, according to the metadata produced at step 3
- save images into subdirectories named after the labels they ere assigned

The result of this process has been uploaded to a public bucket on Amazon S3, from where you can browse and download all files produced in the above described stages.

DATASET CHARACTERISTICS

The dataset contains 51216 images, 320 x 180, in jpeg format, distributed over the following classes. It can be downloaded and extracted from this zip file on S3.

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 of the distribution of the 2 main principal components of the full dataset.

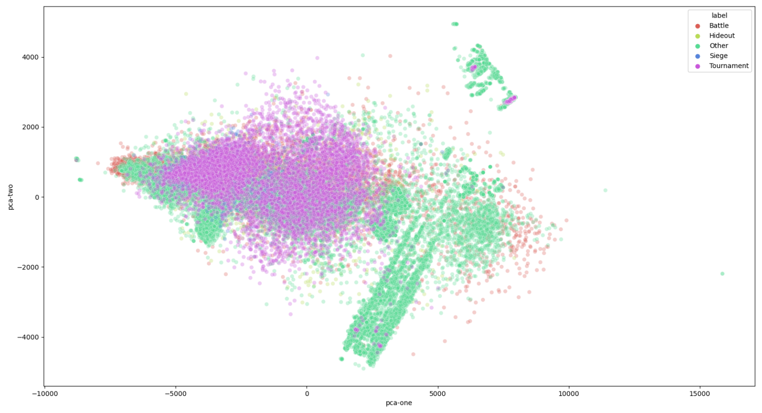


Figure 9: 2 Main Principal Components

and also of the three main principal components, using the same color scheme.

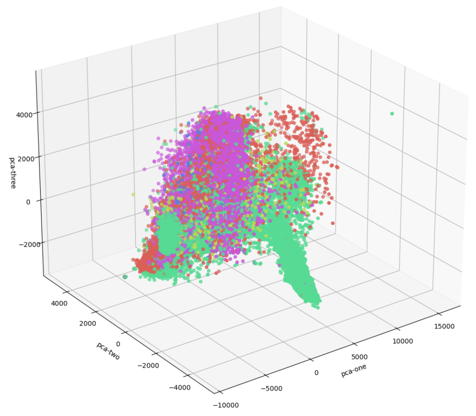


Figure 10: 3 Main Principal Components

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 between themselves and with some of the “Other” images.

Later on, after we have create a VGG13 model, we can plot an alternative PCA representation of the dataset, using the additional generated features that can be recovered from the last layer of the neural net.

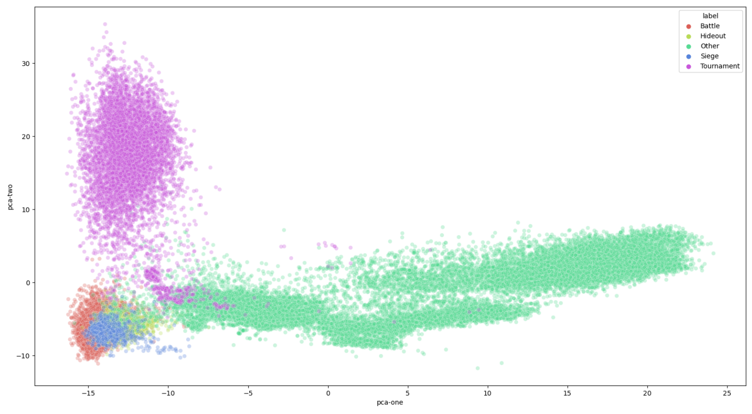


Figure 11: VGG13-2 principal components

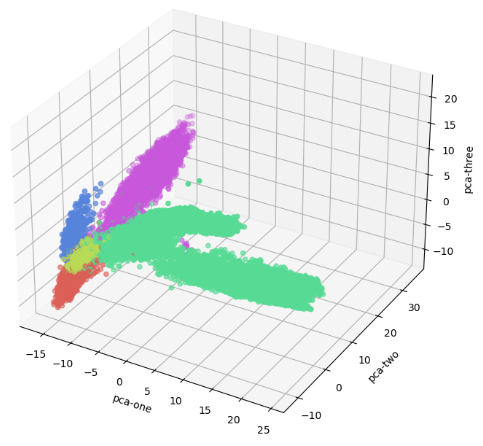


Figure 12: VGG13-3 principal components

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

The script used to produce these visualzations are *pca_vgg.py* and *pca_sklearn.py*

ALGORITHMS AND TECHNIQUES

The general idea is to create first an image classifier to categorize the images extracted from gameplay videos. Then to use the predictions generated by the image classifier as a base for splitting gameplay videos into scenes.

The main approach to build an Image Classifier will be Convolutional Neural Network, since

- this is a well-tried way to approach the problem
- I can use standard CNN topologies available in Pytorch, such as VGG, picking one with the best tradeoff performance/complexity
- at the very least the model would give me more information on what I have to be looking for

A simple to use and flexible topology I decided to use was VGG13, which is powerful enough to give good results 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.

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 picks the Category "Other".

On top of that, I created a very simple model that uses flattened matrixes of images as an input to standard scikit-learn transformers and PCA.

Using greyscale images in 80x45 format, I got the following results using Random Tree Forest and Stochastic Gradient Descent on 100 PCA-produced features, from a 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
```

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
```

METHODOLOGY

DATA PROCESSING

The image dataset has been created extracting frames from videos on youtube, and putting them in subdirectories named like the category they belong 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 without resizing them, 320x180, and in color. This is because this is what the VGG models expect.

As these images originate from video games, any kind of data augmentation such as mirrored or cropped images do not make sense. These artificially produced images would not be produced by the videogame, as for instance many GUI components are always on the same side of the screen.

IMPLEMENTATION

I created scripts and notebooks so that they would work both locally and on Sagemaker. For more details see the file README.md in the Github repository

TRAINING SCRIPT The training script *train.py*:

- uses an image loader from pytorch to create a generator scanning all files in the data directory.
- uses a pytorchvision transformer to resize and normalize images
- splits the dataset in train, validation and test sets, using stratification and shuffling
- loads a VGG13 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 a classification report and a confusion matrix.

VERIFICATION SCRIPT The verification script *verify_model.py* works only locally, as it assumes the model and the dataset has been saved locally from the previous step. This script

- Loads the model created in the previous step
- Walks through all the images in the dataset, one by one, returning a list a predictions (by means of the predictor) and a list of truth values
- prints 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. An examples can be found in *CNN_Fourth_iteration.ipynb*.

REFINEMENT

How the approach evolved can be seen in the Jupyter Notebooks: *CNN_First_Iteration.ipynb*, *CNN_Second_Iteration.ipynb*, *CNN_Third_Iteration.ipynb* and *CNN_Fourth_Iteration.ipynb* (the final one, that can be executed on Sagemaker)

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_F irst_Iteration.ipynb, I had too few examples 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 didn't match with the true values. 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.

After a few tries, 320 x 180 proved also to be the optimal size.

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

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 descriptions, and not create them without human supervision.

To convert image classifications to scenes, I use 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, containing a set of characters representing the initials of the classes the frame is thought to belong to. From these frame representations, scenes in videos are extracted.

The algorithms are in the use *interval* package. Locally I use *predict_intervals_walkdir*, On Sagemaker you can use 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 using). 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. An example of the output:

```
...
41:10  __BBBBBBBBBBBBBBB___
41:12  _BBBBBBBBBBBBBBBBBBB
41:14  _BBBBBBBBBBBBBBBBBBS
41:16  __BBBBBBBBBBBBBBBBBB
41:18  _BBBBBBBBBBBBBBBBBBB
41:20  __BBBBBBBBBBBBBBBBBB
41:22  __BBBBBBBBBBBBBBBBHS
41:24  _BBBBBBBBBBBBBBBBBBB
41:26  _BBBBBBBBBBBBBBBBBBB
41:28  __BBBBBBBBBBBBBBBB_T
41:30  _BBBBBBBBBBBBBBBBB_T
41:32  __BBBBBBBBBBBBBBB__T
41:34  ___BBBBBBBBBBBBBBBHH
41:36  __BBBBBBBBBBBBBBB__T
41:38  ___BBBBBBBBBBBBBBBBH

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 E72, E73, E74 the hero talks a walk around her estates, but the algorithm is confused and hints at some kind of battle, as it is an outdoor scene
- 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
- 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.
- 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 introduce additional categories.

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 in regard to classifying each category. .

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.