CAPSTONE Project - Machine Learning Engineer Nanodegree

October 2020

PROBLEM DEFINITION

BROAD CONTEXT

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 videos 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.

PROJECT LOCATION & FILES

- Main repository on Github: *https://github.com/diegoami/DA_ML_Capstone*
- Companion project: *https://github.com/diegoami/DA_split_youtube_frames_s3.git*
- Data : *https://da-youtube-ml.s3.eu-central-1.amazonaws.com/*

PROJECT SCOPE

The game that I have chosen to analyze is *Mount of Blade: Warband*, of which I made several walkthroughs. In this game, I have identified seven types of scenes to which an image belongs:

- BATTLE: any battle taking place in an open field or in a village
- TOURNAMENT: Tournaments in arena
- HIDEOUT: the warband assaults a bandit hideout
- TRAINING: the hero trains in a field or in a village (later remapped to OTHER)
- SIEGE: a town is sieged
- TRAP: hero falls into a trap and must fight their way out (later remapped to BATLE)
- TOWN (escape): escape from the town or castle prison (later remapped to BATTLE)
- OTHER : everything else

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 66 from following public 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*

These are the episodes I went through and manually split into scenes. I wrote down how they were split in the video description on youtube.

For instance, in episode 54, I have identified following scenes, of the category "Hideout", "Battle", "Tournament", "Town". 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

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

This project: - Downloads 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 an jpeg file, using the opencv python library, resizing to the practical format 320x180 - Download the text from the youtube description and save it along the video (*metadata*) - Distribute the files over directories named by the categories.

This way, I created first a dataset that I uploaded to a S3 bucket: *https://da-youtube-ml.s3.eu-central-1.amazonaws.com/wendy-cnn/frames/wendy_cnn_frames_data.zip* (3.2 Gb) and made public.

PROBLEM ANALYSIS

DATASET ANALYSIS

The amount of images I had generated first, in this way, was 45718, split in eight categories.

This was the breakdown of the images I collected over the 8 classes I mentioned above

- BATTLE: ~13%
- TOURNAMENT: ~14.8%
- HIDEOUT: ~2.5%

- TRAINING: ~1.7%
- SIEGE: ~0.4%
- TRAP: ~0.2%
- TOWN (escape): ~0.2%
- OTHER : ~67.6%

As it can be seen, some categories have few samples, so it was going to be expected that we could have trouble with those A sanity check whether the images are in the correct directory can be done using the notebook *analysis.ipynb*.

For instance battle images should look like that:

Figure 1: Battle images

FIRST ITERATION

Notebook: *Wendy_CNN.ipynb*

The simplest way I chose to verify whether a model is viable was to start and set up a Convolutional Neural Network in Pytorch, as I was pretty sure that

- this was pretty much the most sensible way to approach the problem
- I could use standard CNN topologies available in Pytorch
- analyzing the result of the model would give me more information on what I would have to be looking for

Convolutional Neural Network are, as a matter of fact, a very standard approach for categorizing images. A simple to use and flexible topology I decided to use was VGG, which is included in the Pytorch library.

As the images extracted from game walkthrough are not related to real world images, using a pretrained net and expand it with transfer learning did not seem to make sense. Instead, I opted for a full train. In the preprocessing phase, in this iteration, I resized images to 128 x 72, which should be enough for the algorithm to recognize features (original images were all 640 x 360). As I already have over 45000 images, I thought I would do not need any kind of data augmentation (like, use mirrored images), also because it is not given that the game may actually show mirrored images.

The flaw in the dataset, regrettably, is that some categories, such as *Siege*, *Trap* and *Town*, have relatively few samples. Looking at the confusion matrix I got, there were however some surprises in the report I got when I decided to test the produced model with 10% of the dataset.

Confusion Matrix

X	0	1	2	3	4	5	6	7
0	5671	9	93	1	19	0	1	0
1	59	1041	49	1	3	0	0	0
2	315	19	30312	1	156	0	77	0
3	26	2	2	164	0	0	0	0
4	67	7	290	1	6401	0	4	0
5	17	0	40	0	0	3	0	0
6	8	1	175	0	49	0	537	0
7	13	1	62	0	1	4	0	8

class	class_name	precision	recall	f1-score	support
0	Battle	0.92	0.98	0.95	5974
1	Hideout	0.96	0.90	0.93	1153
2	Other	0.98	0.98	0.98	30880
3	Siege	0.98	0.85	0.91	194
4	Tournament	0.97	0.95	0.98	6770
5	Trap	0.43	0.05	0.09	60
6	Training	0.87	0.70	0.77	770
7	Town	1.00	0.09	0.16	89
avg	0.89	0.69	0.72	45710	
weighted avg	0.97	0.97	0.96	45710	

with accuracy of 0.97% (regrettably not relevant)

It turned out that the *Siege* class is not that big a problem (as a matter of fact, images belonging to this category are pretty distinctive). However, the classes *Trap*, *Town* and *Training* tended to be misclassified often. After checking the confusion matrix, I decided that it would make sense to remove these three categories, so that the category *Training* is classified as Other (Training is not interesting anyway) while Trap and Town are classified as Battle.

SECOND ITERATION

First, I make sure to create a second dataset, where I map Trap and Town to Battle, and Training to Other. So that I end up with 5 categories:

- Battle : 5943 (13.0%)
- Hideout : 1153 (2.5%)
- Other : 31650 (69.2%)
- Siege : 194 (0.4%)
- Tournament : 6770 (14.8%)

TOTAL : 45710

I chose a smaller format for the images I save, as 640x480 is too big for any model I can realistically train. The dataset becomes therefore much smaller: 1.0 Gb and can be found at *https://da-youtube-ml.s3.eu-central-1.amazonaws.com/wendy-cnn/frames/wendy_cnn_frames_data_2.zip*

I also added a preprocessing step to correct some of the wrongly classified images that are in the dataset, and that I discovered using a simple self-made tool.

Now, creating a basic VGG net (type B) on the full images, having image_height x image_width = 160 x 90, with 5 epochs, and just 5 categories, and then running the model on the full dataset, gives this result:

confusion Matrix

X	0	1	2	3	4
0	5878	6	44	6	9
1	7	1131	3	9	3
2	299	23	31231	3	94
3	1	1	1	191	0
4	10	30	171	3	6586

class name	class	precision	recall	f1-score	support
Batle	0	0.95	0.99	0.97	5943
Hideout	1	0.97	0.98	0.98	1153
Other	2	0.99	0.99	0.99	31650
Siege	3	0.91	0.98	0.95	194
Tournam	4	0.98	0.97	0.98	6770
macro avg	0.96	0.98	0.97	45710	
weighted avg	0.99	0.98	0.98	45710	

with accuracy of 0.98 %

which is a much better result than the first run. I decided that I could keep this model.

IMPLEMENTATION

I set up scripts and notebooks so that they would work both locally and on Sagemaker. However, some things work better locally, while some other work better 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 code root directory is letsplay_classifier - scripts should be executed from this directory, or the directory should be included in PYTHONPATH.

REQUIRED ENVIRONMENT VARIABLES

All scripts require following environment variables, which are the ones required by Sagemaker containers.

- SM_CHANNEL_TRAIN: location of data - the directory where you unzipped the required data
- SM_MODEL_DIR: where to save the model
- SM_HOSTS: should be "[]"
- SM_CURRENT_HOST: should be ""

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

These are the steps that are executed:

- use an image loader from pytorch to create a generator scanning all files in the data directory. This works only if data is local and not on Sagemaker, for which I have to update the dataset.
- use a pytorchvision transformer to resize images
- divide the dataset in train and validation sets, using stratification and shuffling
- load a VGG neural network, modified so that the output layers produce a category from our domain (5 in total 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

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, and a confusion matrix

MISCLASSIFIED IMAGES

I found out that there were images in the training / validation set that were misclassified, therefore I thought about correcting the dataset using a GUI. It would have gone through images that have a very high probability of being classified wrongly, and save the images where I reject the expected label for the predicted one under *rejected.json*. I dropped this approach and it turned out require a lot of overhead and was error-prone

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 JSON payload to a Pillow Image
- model_fn: predicts a category using a previously trained model, from an image in the domain space (a screenshot from Mount Blade in format 320 x 180)
- output_fn: returns the model output as a list of log probabilites for each class

ENDPOINT

The file *endpoint.py* contains a method to call an endpoint with a subset of model and collect predictions, 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 serves a similar purpose.

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

JUPYTER NOTEBOOKS

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

- analysis.ipynb: just to analyse data
- 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 correction and verification how the model splits videos

RESULTS

IMAGE CLASSIFICATION

The classification report and confusion matrix of the finally selected model are shown above.

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. Therefore I created two *intervals predictor* that I could use locally (*predict_intervals_dataloader* and *predict_intervals_walkdir*, and one that I could use on Sagemaker: *predict_intervals_endpoint*) .

The approach I use in this script is to ignore one or two frames that are wrapped inside a scene. Moreover, long sequences of frames that are not classified as "other" (battles, sieges, tournaments, hideouts) are clumped together, as they can sometimes be confused with each other. Therefore, I I chose the next episode in the playlist, E67, and this was the result:

INTERVAL	PREDICTION	REALITY
23:04-28:04	Hideout : 18% , Other : 6% , Siege : 72%	Siege of Unuzdaq Castle
46:24-47:44	Battle : 68% , Other : 19% , Tournament : 10%	Battle with Desert Bandits
52:30-53:02	Battle : 54% , Hideout : 26% , Other : 16%	Trap in Dirigh Abana (Battle)
54:38-56:00	Battle : 77% , Other : 10% , Tournament : 12%	Battle with Boyar Gerluch

INTERVAL	PREDICTION	REALITY
01:03:52-01:05:42	Battle : 35% , Other : 13% , Tournament : 42%	Battle with Steppe Bandits (knockd out)
01:14:00-01:16:36	Battle : 75% , Other : 8% , Tournament : 15%	Battle with Emir Atis
01:17:50-01:19:16	Battle : 83% , Other : 10%	Battle with Emir Hamezan
01:33:12-01:34:22	Battle : 93%	Battle with Emir Rafard
01:38:18-01:43:50	Battle : 64% , Other : 8% , Tournament : 26%	Battle with Emir Dashwhal (1)
01:43:54-01:46:06	Battle : 56% , Other : 7% , Tournament : 36%	Battle with Emir Dashwhal (2)
01:49:00-01:50:38	Battle : 91% , Other : 6%	Battle with Emir Ralcha (1)
01:50:50-01:53:32	Battle : 94%	Battle with Emir Ralcha (2)
01:55:52-01:57:46	Battle : 88% , Other : 6%	Battle with Emir Azadun

Compare this with the actual transcript I created after watching the video

CONCLUSIONS

This project proved to me that it is possible to reliably build a classification model for images. I could apply this technique also to other video games, as the requirement of splitting game walkthroughs in scenes is something that is common over many games.

It als proved that this model can be used to successfully split videos into scenes, with some postprocessing.