(Not really) achieving state-of-art classification accuracy in video classification

Álvarez J. Sáez J.

I. INTRODUCTION

It is known that the supervised image classification problem is getting close to be completely solved [3]. The developed models are becoming more complex and powerful, being able to obtain very good representations that lead to models with incredible performance.

Videos are concatenations in the time axis- of numerous images, where it is common that the image in time t is related to images $t-1$ and $t+1$, and probably also related to further steps of time. With this consideration, it is natural to extend the techniques used in single images to videos in order perform different tasks.

In this practical assignment, we consider the video classification task. In this task, given a set of videos $\mathscr{X}$ labeled by **actions performed in this video** $\mathscr{Y}$, we aim to find a function $f : \mathscr{X} \to \mathscr{Y}$ that tells which action is performed in each video of our set $\mathscr{X}$. This is also known as the **action recognition task**.

To achieve this, we will test a few different methods: a fixed class one, a random class one, and a CNN frame-by-frame model. We will present a theoretical analysis of the error of the first two methods and then explain how we executed the third method and the results obtained. We obtain very high accuracy scores when using the frame-by-frame method. These accuracy scores decrease when we increase the number of classes.

The rest of this report is structures as follows: section II briefly describes the implemented methods, chapter III discusses implementation details, chapter IV describes the dataset used, chapter V details our models results and the respective analysis, chapter VI presents our final conclusions and chapter VII describes the time log followed for this assignment.

II. METHODOLOGY

In this section we briefly describe the video classification models implemented for this practical assignment and develop a theorical analysis for the results expected by two of them.

Let us stablish a simple notation for our theorical study. Let M be the number of classes in our dataset and let $X\{(x_i, c_i),\ i = 1, \ldots, N\}$ our dataset, where c_i is the class associated to sample i. Our guess for the $i-$sample will be $\hat{c}_i$, chosen differently depending on the method. Our main metric will be the **accuracy**:

$$\textbf{Acc} = \mathbb{E}\left[\mathrm{P}\left[c_i = \hat{c}_i\right]\right].$$

Finally, let $|C_j|$ be the number of elements in the $j-$th class, for $j \in \{1, \ldots, M\}$. Since every sample has a single associated class we also know that

$$\sum_{j=1}^{M} |C_j| = N. \tag{1}$$

A. Method 1: Fixed class

The first model implemented is quite simple: a single class is fixed beforehand, and the model predicts that class for every sample. Let us study the expected accuracy for the method.

Let $m \in \{1, \ldots, N\}$ be our fixed class. Then $\hat{c}_i = m$ for every sample. The accuracy can be expressed as follows

$$\textbf{Exp Acc} = \mathbb{E}\left[\mathrm{P}\left[c_i = m\right]\right].$$

If the dataset was balanced (there were the same number of samples in every class), the probability of element c_i being in class m would be $1/M$:

$$\textbf{Exp Acc}_{\text{bal}} = \mathbb{E}\left[\mathrm{P}\left[c_i = m\right]\right] = \mathbb{E}\left[\frac{1}{M}\right] = \frac{1}{M}.$$

However, this is not always the case. For instance the dataset used in this practical assignment, **UCF101**, is not balanced, as we will see in section IV. In this case, the probability of the $i-$th sample being in class m is the ratio of elements in the m class in our dataset:

$$\textbf{Exp Acc}_{\text{inbal}} = \mathbb{E}\left[\mathrm{P}\left[c_i = m\right]\right] = \mathbb{E}\left[\frac{|C_m|}{N}\right] = \frac{|C_m|}{N}.$$

Finally, the inbalanced case must generalize the balanced one. If the data was balance, the number of elements in the m class would be N/M, and the balanced and inbalanced accuracies match:

$$\textbf{Exp Acc}_{\text{inbal}} = \frac{|C_m|}{N} = \frac{N/M}{N} = \frac{1}{M} = \textbf{Exp Acc}_{\text{bal}}.$$

B. Method 2: Random choice

The second model is not much more complicated: for every video the model predicts a class chosen uniformly at random. Let us study the expected accuracy for the method.

In this case, our prediction $\hat{c}_j$ is uniformly randomly pick from the set {1, ..., M}. The accuracy can be computed directly:

$$
\begin{aligned}
\textbf{Exp Acc} &= \mathbb{E}\left[\mathrm{P}\left[\hat{c}_i = c_i\right]\right] \\
&= \mathbb{E}\left[\sum_{m=1}^{M} \mathrm{P}\left[\hat{c}_i = m\right] \mathrm{P}\left[c_i = m\right]\right] \\
&= \mathbb{E}\left[\sum_{m=1}^{M} \frac{1}{M} \frac{|C_m|}{N}\right] \\
&= \mathbb{E}\left[\frac{1}{MN} \sum_{m=1}^{M} |C_m|\right] \\
&\overset{(1)}{=} \mathbb{E}\left[\frac{1}{MN} \cdot N\right] \\
&= \frac{1}{M}
\end{aligned}
$$

where in (1) we used equation 1. As we can see, the way of selecting a random class for each sample makes the expected accturacy stable so it doesn't depend on the number of classes. If we were to choose between this and the previous method without previously knowing the distributions of classes in the dataset, the second algorithm would be more consistent.

C. Method 3: CNN

This method is the first method that is not completely doomy. As we mentioned in the introduction, videos are concatenations of images so a very good idea is to treat them as splitted images.

In this assignment, we will use the frame-by-frame paradigm. In particular, we will use the framework proposed in [4].

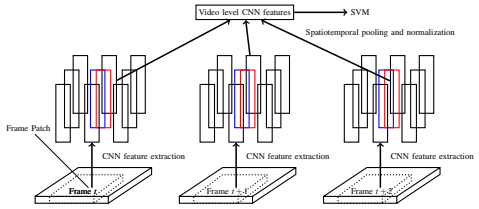

Fig. 1: Overview of the proposed video classification pipeline.

A brief summary of the framework's workflow is shown in Figure 1. The main idea is to split the whole video in different frames and apply a CNN to each of the images and, when the features are extracted, perform an spatiotemporal pooling to summarize the features of all the frames into a single vector that can be classified using any kind of model. This method acheived state-of-the-art results in UCF101 when it was firt presented.

It is important to remark that, in this assignment, the CNN architecture that we use is InceptionV3, which can be automatically loaded from Keras. We substract the last layers and manually add the spatiotemporal pooling and a linear classifier.

In this case, computing the expected accuracy of our model is not straightforward, since it is a neural network [1].

III. IMPLEMENTATION

A. Functions for Methods

The first two methods are implemented in file `random_vs_fixed_mode.py`. The fixed class for the first method is hardcoded in the file, and the implementation itself is quite straight-forward.

The third model is implemented in file `random_vs_fixed_mode.py`, specifically in the `get_model()` function. We only needed to obtain the InceptionV3 pretrained model without the last top layers, add a global average pooling and a final dense layer.

Additionally, functions `freeze_all_but_top()` and `freeze_all_but_mid_and_top()` were implemented. They freeze every layer but the top two and every layer but the midel and top layers respectively, and are used during training. Firstly, the top two layers are trained with the default learning rate, 0.01. Secondly, the mid and top layers and trained with a much lower learning rate, 0.0001, for fine-tuning.

B. Checkpoints

When it came to executing the experiments of the last model, we struggled executing it for the whole 100 epochs. The execution of the training in the file `train_cnn.py` failed after an arbitrary number of epochs (about thirty: sometimes more, sometimes less). The error shown was:

```
Error occurred when finalizing GeneratorDataset
iterator: FAILED_PRECONDITION: Python interpreter
state is not initialized.
The process may be terminated.
         [[{{node PyFunc}}]]
```

After a deep search in different forums (such as StackOverflow, Github and Keras documentation), we could not determine the specific reason for this mistake, although we had the suspicion that the error was dued to a difficulty in matching the batch size with the steps per epoch [1].

We decided to, since everything during the training seemed to be working fine, implement a few functions to be able to resume the training at any point. This way we only would have to reexecute the train script. In order to achieve this, we added the following functionalities:

- Changed the folder structure in order to have all the logs of the same model in the same file (a log file is created for each execution of the train script, we merged them all to create the results). The new structure is `data/logs/'class_limit'_'seq_limit'`. Using this, we got our results well split and plotting the results only requires file management.
- A new `ModelCheckpoint` that creates a checkpoint every 10 epochs.
- A function that loads the last checkpoint. This function is called `load_previous_weights` and, briefly, it

[1] According to this answer from Stack Overflow

searches the required weights file matching the number of classes and segment size and loads it (if it exists) and returns the number of executed epochs. Using the number of epochs, the model only trains the remaining epochs up to 100.
- Lastly, the pipeline for beginning the training is changed a little bit so that we firstly try to load a previous model automatically. Thanks to this, if our script fails, we only have to reexecute the script to continue the training.

Thanks to this functionality, we were able to finish this assignment. Otherwise the code could not have been executed.

C. Running the code

The practical assignment submission consists of two jupyter notebook called `VideoClassification.ipynb` and `plotting.ipynb`, along with a folder called `P2`. The experiments can be reproduced by following step by step the jupyter notebook, although the data and the saved checkpoints are not in the submitted folder.

Along the execution, several checkpoints will be automatically created. As explained in the proevious subsection, the code will automatically load the last saved checkpoint and continue the execution from that point onwards.

After the execution, we can use the obtained logs and the functions from file `plot_utils.py` (adapted from the one given provided for this assignment) to plot this accuracy and loss evolutions. This process is automated in the jupyter notebook `plotting.ipynb`.

Finally, the different arguments and parameters and in the relevant implemented functions. For any changes, see the jupyter notebook or the respective python files.

IV. DATA

In this section, we will describe the used dataset. UCF101 [2] was (when first presented) the largest dataset of human actions. As it name indicates, it consits on 101 action classes. In this dataset, we can find over $13k$ clips and 27 hours of video data.

Fig. 2: A few examples of labeled actions from the dataset UCF101 [2].

In the first experiment, we used only the first 5 classes of the dataset. In Figure 3 the histogram of the class distribution is shown. As we can see, these five classes are a little bit imbalanced, which will lead to not exactly uniform results in the random algorithms, as we explained in the theoretical disquisition from section II.

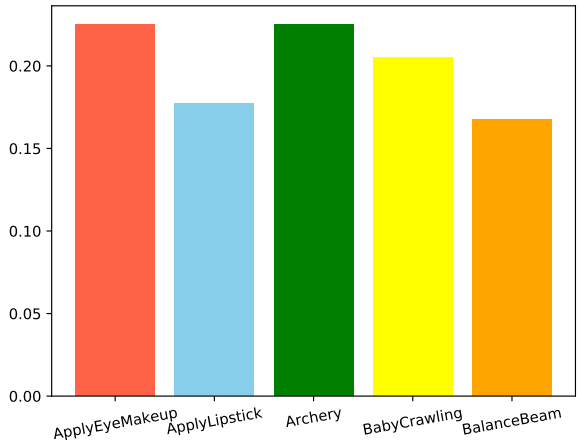

Fig. 3: Histogram of number of examples per class in UCF5.

For the later experiments, we used the first $10, 15$ and 20 classes respectively, taken in an alphabetical order from the complete dataset. Also, with respect to the train and test partitions, they were created using the scripts provided for this assignment. These scripts divide the data in train and test (script `1_move_files.py`) using pre-made train and test lists created in the documentation of the dataset. Also, all the videos are divided in frames and stored in train and test folders automatically using the script `2_extract_files.py`.

V. RESULTS AND ANALYSIS

A. 5-classes experiments

In order to test the fixed and random class models we repeated the validation process 20 times. Table I shows the min, max, mean and standard deviation of the accuracy for both models.

	min	max	mean	std
Fixed mode acc	0.2252	0.2252	0.2252	0.0000
Random mode acc	0.1832	0.2376	**0.2048**	0.0156

TABLE I: Results of the dummiest models.

We can see how the fixed-class model's accuracy doesn't change over iterations, as expected. For these experiments, the first class is fixed for the mixed model. That is why the accuracy matches the ratio of elements in the first class over the whole dataset.

On the other hand, the random accuracy is not as consistent, and the obtained mean is quite close to the expected value, $1/5$, since there are 5 classes in our dataset. Both results matches our theorical analysis.

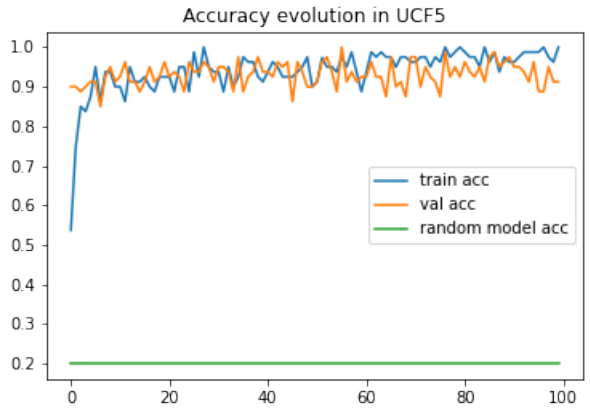

Fig. 4: Train and validation accuracy curves of the CNN framework in the UCF5 dataset, compared to the dummy models.

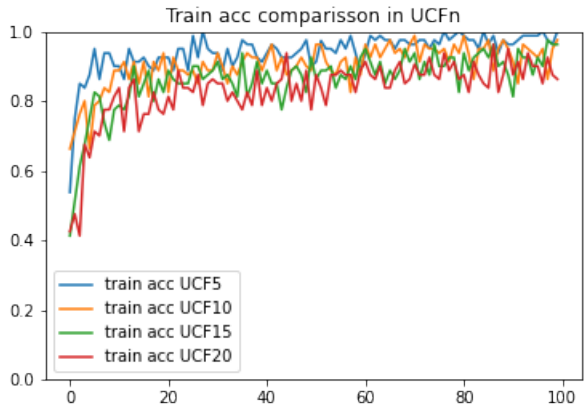

Fig. 5: Accuracy of the different models in the train set.

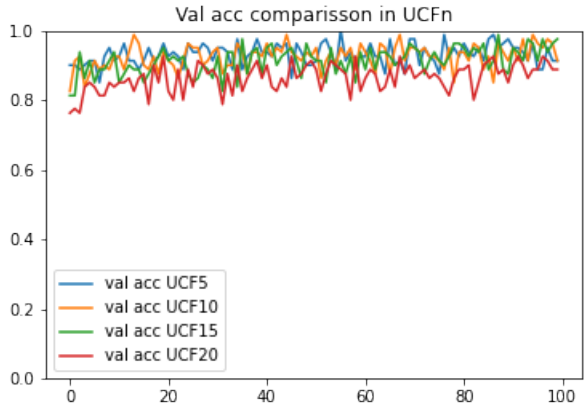

Fig. 6: Accuracy of the different models in the validation set.

Figure 4 shows the comparison of the frame-by-frame framework with the previous models. The CNN model clearly outperforms the random model, which was expected. With respect to the CNN framework results, we can say that the model achieves a really good accuracy score in about 15 epochs and then the improvement is not very impressive. Following our intuition, the model should Early Stop. In fact, we checked that the model has a `CallBack` with the early stopping with a period of 10 epochs. However, we state that the oscilations of the learning curves (the loss function can be observed in Figures 7 8) lead to avoid the early stop.

Model	Accuracy
Fixed mode acc	0.2252
Random mode acc	0.2048
CNN	**1.000**

TABLE II: Comparison of the models used for UCF5.

Table II depicts the comparison between the first two models and the CNN framework. We obtain a really odd result: the CNN method obtains the highest possible accuracy 1.0 in the train and validation set. This is an indicator that something is probably wrong. However, this is not a part of our developed code and we could not quickly determine the exact reason of this result.

B. Complexity evolution with more classes

The following goal is to increment the number of classes gradually and compare how the model performs with this class increment. A priori our model should decrease its performance when the number of classes increases since the problem becomes harder to solve, so it is harder for the models to generalize.

Figures 5, 6 show that our previous statement is true. As we can see, the model that consider the lesser number of classes, called UCF5, achieves the highest accuracy values in each of the epochs. Also, each increment of the number of classes leads to a small decrease in the accuracy values, being the model that considers 20 classes (UCF20) the one with the lowest results in terms of accuracy.

Figures 7 and 8 show the exact same behaviour when we consider the loss function. The higher the number of classes considered, the higher loss values obtained.

Additionally and although we do not specifically show any data we detected that the training time slightly increases when incrementing the number of classes.

Finally, table III summarizes the results obtained for each of the different number of classes. These results match the previous analysis. It is remarkable the very high accuracy that this method obtains then the number of classes is as limited as ours. It could be studied if, in this case, the accuracy decreases linearly with the number of classes.

	max train acc	max val acc	min train loss	min val loss
UCF5	1.0000	1.0000	0.0439	0.0593
UCF10	0.9875	0.9875	0.1279	0.1042
UCF15	0.9750	0.9875	0.2339	0.1190
UCF20	0.9625	0.9250	0.3759	0.2504

TABLE III: Results of the different UCFn models.

We have also noticed that all the values obtained are

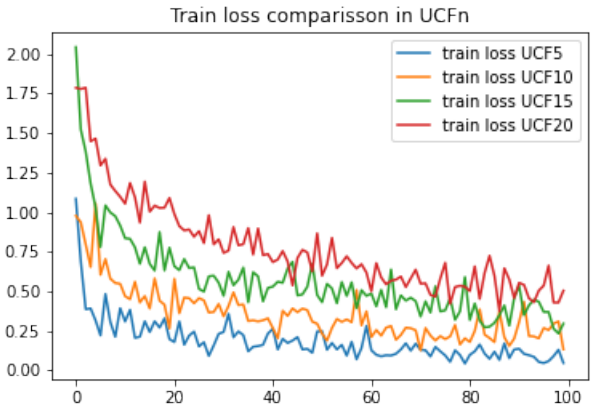

Fig. 7: Loss of the different models in the train set.

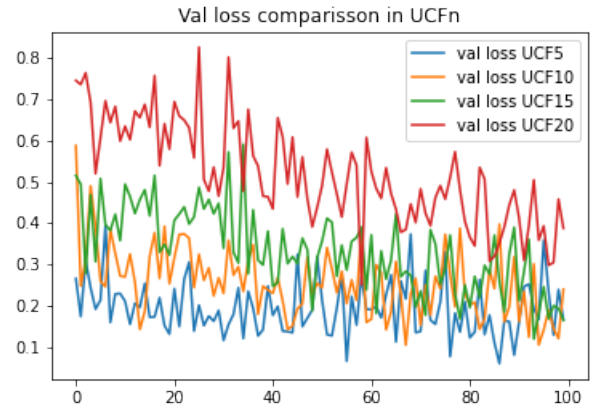

Fig. 8: Loss of the different models in the validation set.

multiples of $0.005 = 1/200$, which is precisely the inverse of the batch size. Because of this, our current hypothesis is that the average accuracy is computed per batch instead of per epoch but we haven't been able to confirm nor deny such hypothesis. It is worth mentioning that we have not modified that part of the code.

VI. CONCLUSIONS

In this practical assigment we have tackled the **action recognition task**, using one of the most important datasets for this problem with very advanced and complex (and not so complex) methods. These models perform really well when the number of classes is limited, and struggle generalizing when the number of classes increases. This behaviour depicts that for the whole dataset (101 classes) we might need state-of-the-art models such as transformers [5].

Despite the problems with Google Colab during the experiments of this assigment, we have learned a lot and genuently enjoy this practical assigment.

VII. TIME LOG

This practical assignment was fulfilled over the course of two full days, where both members of the team worked (literally) side by side with the following time log:

- Theorical analysis: 2h.
- Implementing the models: 1h.
- Making the third model work on Colab: 10h.
- Executing the experiments: 2h.
- Analyzing and explained the results: 2h.
- Writing the final report: 8h.

The model trainings was parallelized with the report writing due to the long exeution times.

REFERENCES

[1] Jakob Gawlikowski et al. *A Survey of Uncertainty in Deep Neural Networks*. 2021. DOI: 10.48550/ARXIV.2107.03342. URL: https://arxiv.org/abs/2107.03342.

[2] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. *UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild*. 2012. DOI: 10.48550/ARXIV.1212.0402. URL: https://arxiv.org/abs/1212.0402.

[3] Jiahui Yu et al. *CoCa: Contrastive Captioners are Image-Text Foundation Models*. 2022. DOI: 10.48550/ARXIV.2205.01917. URL: https://arxiv.org/abs/2205.01917.

[4] Shengxin Zha et al. *Exploiting Image-trained CNN Architectures for Unconstrained Video Classification*. 2015. DOI: 10.48550/ARXIV.1503.04144. URL: https://arxiv.org/abs/1503.04144.

[5] Hao Zhang, Yanbin Hao, and Chong-Wah Ngo. "Token shift transformer for video classification". In: *Proceedings of the 29th ACM International Conference on Multimedia*. 2021, pp. 917–925.