Neural Networks: Implementing a Convolutional Neural Network with Keras for action recognition in videos Deep Learning in Computer Vision Lab 5

Awet Haileslassie Gebrehiwot, Santiago Herrero Melo

I. INTRODUCTION

In this fifth lab session of *Deep Learning in Computer Vision* we will see how to build a convolutional neural network with the Keras framework in Python aimed at recognizing actions in videos. We will use the UCF101 dataset, which is formed by YouTube videos gathered in 101 categories.
In this lab session, we will work with the first 30 categories of videos, being each of them composed by three videos, two of which will be used as training set, and the other one as validation set.
A Convolution neural network(CNN) composed of the following components:

- *Convolutional layers*
- *Pooling layers*
- *Fully connected layers*
- *A choice of activation function and optimizer*

II. IMPLEMENTATION

In order to classify the actions, we will use a siamese convolutional network. This network is called siamese because it is formed by two branches that uses the same weights Each branch has the same architecture while working in tandem on two different input vectors to compute comparable output vectors. One of the branches will correspond to the actual frame of the video (RGB frame), while the other one will correspond to its optical flow. In this way, by checking both characteristics at the same time and merging them, we will obtain the probabilities of the video belonging to each of the 101 classes.

A. Extract RGB

The first thing to do is to obtain RGB content for each video sequence. For doing so, we use from the provided code the functions *extract_RGB*, where we can extract RGB information from the video content.

B. Calculating optical flow

The 2nd thing to do is to compute flow for each video sequence. For doing so, we use from the provided code the functions *compute_flow*, where we can calculate and compute the optical flow from the video.

C. Creating the branches

For creating the neural network, we will use the *functional API* architecture in Keras, in which each part of the network is an input for the following part.
Each part will consist of three pairs of convolutional layer plus max pooling, being the number of filters in each layer 30, 60 and 80. For each convolutional layer we will use a kernel of size 3, padding and the *ReLu* function as activation function.
After these layers, we will flatten the result before connecting to the hidden layer (which will have 500 filters) and the output layer, which has 30 outputs and uses a *softmax* function.
Finally, the model is compiled with *categorical cross entropy* and a *sgd* optimizer, with certain specific parameters that are already given.
Both for the frame branch and the flow branch, the structure of the branch is the same. Therefore, the code is also the same, and when running one or the other, we just have to change the parameter *data_type*.
In the main script, we have to load the pre-trained weights. Once that is done, we just have to call the function *evaluate_generator* in exercise 1, and *fit_generator* in exercise 2. For using these functions, a generator is created for the training and the validation set, using the already provided *My_Data_Sequence_one_branch* function. In the case of *fit_generator*, we use the validation set as validation set. In the case of exercise 2, some callbacks are provided in order to visualize the results in TensorBoard.

D. Creating the siamese network

For creating this network, we will use the same structure as in the previous case. This time, we create an alternative function in the *model* module, called *make_branch_model*. In this case, we just copy the structure of the *make_one_branch_model* function until the creation of the hidden layer, which will be the output of the function.
The *make_branch_model* function is called twice inside the *make_model* function, once for the frame branch (channels = 3) and once for the flow branch (channels = 2). After this, we use the *concatenate* function with 1000 filters. Afterwards, we will just add the output layer, which will have 30 filters (as many as video categories we are working with).

The siamese network is shown in Fig. 1.

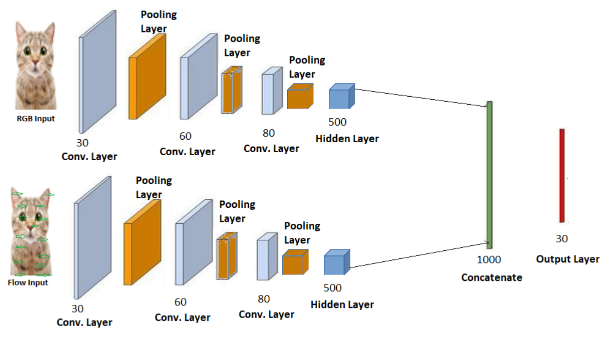


Fig. 1: Built siamese network

E. *Running the code*

For running the code we first have to activate the Tensorflow environment. Once that is done, we access the folder where the skeleton of the code is saved, and type the following line in the command window:
python lab5_0.py in order to get the flow of each of the videos,
python lab5_1_skeleton.py in order to obtain the accuracy of the test set, each for the frame and the flow separately.
python lab5_2_skeleton.py in order to obtain the accuracy of the train set, each for the frame and the flow separately.
python lab5_3_skeleton.py in order to obtain the accuracy of the train set for the full siamese network.
For visualizing the results in TensorBoard for exercises 2 and 3, you must type in the command window

tensorboard –logdir=TensorBoard
and then copy the obtained URL in an internet navigator that is not Firefox.

III. RESULTS AND ANALYSIS

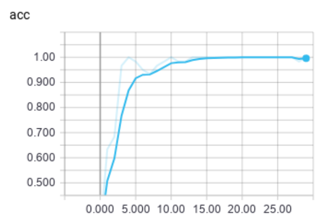
A. *Exercise 1*

In this exercise, we obtain the accuracy's of the test set each for the frames and for the flow.
In the case of the RGB frames, the accuracy is *0.4333333*.
In the case of the flow, the accuracy is *0.2333333*.

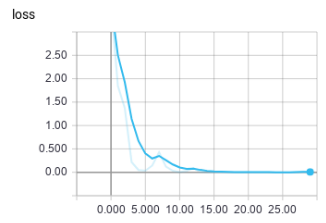
B. *Exercise 2*

For this exercise, we obtain the results of accuracy and loss function for both training and validation set, for the RGB frame model and the flow model, respectively. In the case of the frame model, we obtain the graphs shown in Fig. 2.

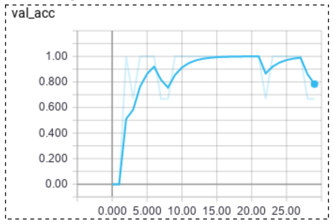
In Fig. (2a) we can see how, after 15 epochs, the branch reaches an accuracy of 100%, and in Fig. (2b) it is shown how after the same number of epochs, the loss function is zero.
On the other hand, for the validation set, the accuracy falls a



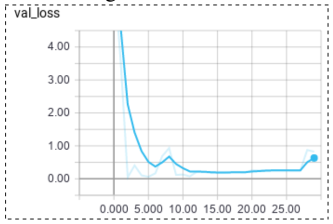
(a) RGB branch Accuracy of the training set



(b) RGB branch Loss function of the training set



(c) RGB branch Accuracy of the validation set

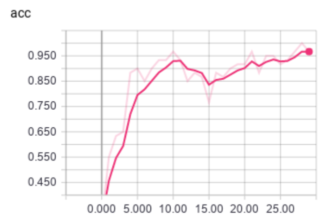


(d) RGB branch Loss function of the validation set

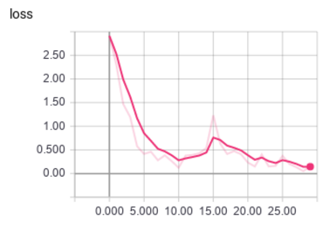
Fig. 2: Accuracy performance and loss function for the RGB frames branch

bit after 21 epochs, to reach again 100% accuracy and finally fall again after 27 epochs, ending with around an 80% of accuracy, as shown in Fig. (2c).

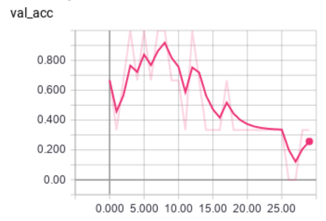
For the flow model, we obtain the graphs shown in Fig. 3.



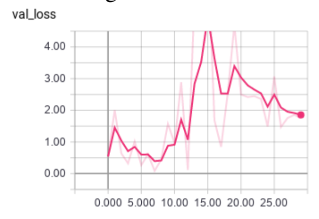
(a) Flow branch Accuracy of the training set



(b) Flow branch Loss function of the training set



(c) Flow branch Accuracy of the validation set



(d) Flow branch Loss function of the validation set

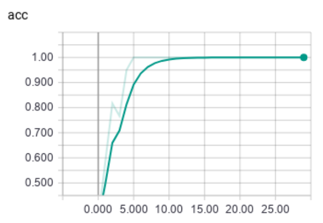
Fig. 3: Accuracy performance and loss function for the Flow branch

We can see enormous differences between the performance of the accuracy in the train set (Fig. (3a)) and the validation set (3c)). In the first case, the accuracy grows rapidly until the eleventh epoch, when it drops slightly for a few epochs, to finally grow gradually, achieving a value over *0.95*. For the validation set, it also grows rapidly reaching high values, but after the twenty ninth epoch it decreases, with some peaks, arriving to a final value of *0.28*.

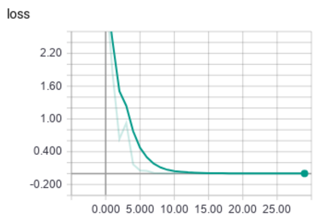
The loss functions behave as a consequence of these graphs, being in the case of the validation set very high in some points, as shown in Fig. (3d).

C. Exercise 3

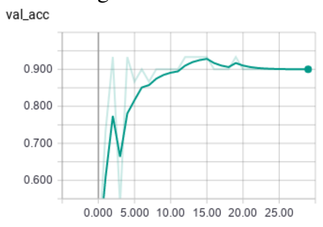
In this exercise, we have used the siamese network, obtaining the results shown in Fig. 4.



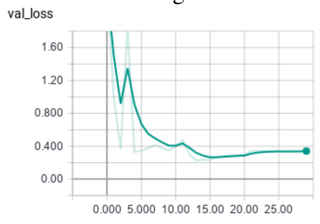
(a) siamese network Accuracy of the training set



(b) siamese network Loss function of the training set



(c) siamese network Accuracy of the validation set

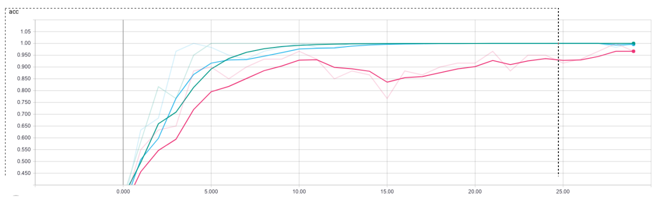


(d) siamese network Loss function of the validation set

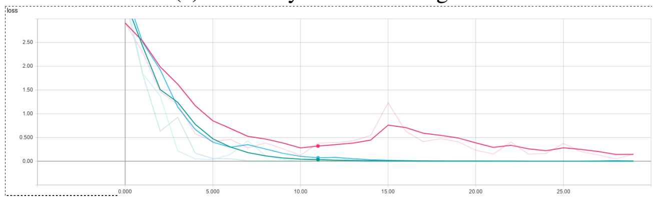
Fig. 4: Accuracy performance and loss function for the siamese network

In Fig. (4a) we can perceive how the network rapidly reaches a 100% accuracy, staying in that value until all the epochs are run. In the case of the validation set, it also reaches stability very fast, but in the value of *0.9*, as shown in Fig. (4c).
As a consequence of these accuracy's, the loss functions for both cases reach very low values, being even *0* in the case of the training set (Fig. (4b).
In order to understand better the improvement achieved by concatenating the frame and the flow branches, this is, by creating the siamese network, we have Fig. 5, where the blue lines correspond to the frame branch, the red lines correspond to the flow branch, and the green lines correspond to the siamese network.

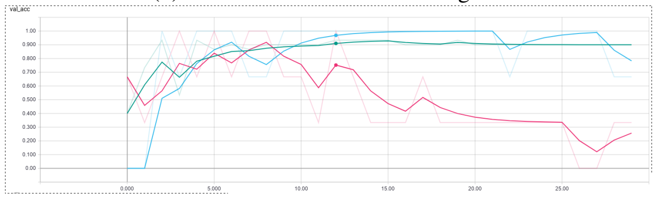
As we can observe from the above Fig.(5), The siamese networks which combines both single branched (RGB and Flow) out-performs the single RGB based as-well as Flow computation. The Resulting accuracy is also stable as compared to the Single branched RGB as-well as Flow. This is due-to the fact the siamese network combines both features to maximize the accuracy and minimize loss.



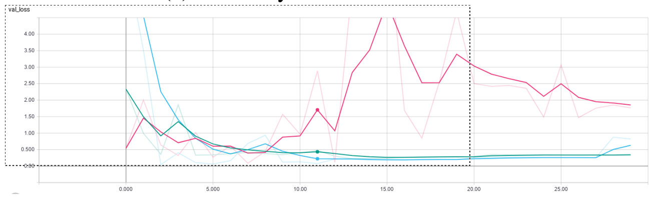
(a) Accuracy of the training set



(b) Loss function of the training set



(c) Accuracy of the validation set



(d) Loss function of the validation set

Fig. 5: Accuracy performance and loss function for the siamese network(Light green) vs Single branch RGB(light Blue), and Flow (light red)