INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact factor: 6.078
(Volume 6, Issue 3)
Available online at: www.ijariit.com

Duration of an actor in a video using Keras

Puja Kedia
pujakedia111@gmail.com
Sir M. Visvesvaraya Institute of Technology, Bengaluru, Karnataka

Saurabh Singh
saurabhsingh183183@gmail.com
Sir M. Visvesvaraya Institute of Technology, Bengaluru, Karnataka

Lakshmi Saai Rasazna Konagalla
klsrasazna@gmail.com
Sir M. Visvesvaraya Institute of Technology, Bengaluru, Karnataka

Shantha H. Biradar
shantha_is@sirmvit.edu
Sir M. Visvesvaraya Institute of Technology, Bengaluru, Karnataka

ABSTRACT

This research is mainly based on automating the process of calculating the time taken for any actor to appear on the screen. It is one of the main factors for determining the remuneration to be given to actors for appearing on the screen for a particular time period. By automating this process, we can accurately determine the screen time of an actor with minimum error. This proposed idea can be implemented using the knowledge related to image processing with the help of CNN architecture. The major part of the research lies in determining the hyperparameters and the right model that fits the given video appropriately and gives the best results for the model evaluation. The major findings of this paper lie on analyzing the right activation function, the number of layers for the neural network, finding the drop-out rate for the trained neural network, deciding upon the weight sharing of the input attributes and of the hidden layers. The final outcome of this performed experiment is a neural network which can be used for deciding the duration of an actor on the screen.

Keywords— *CNN, Drop-Out Rate, Keras, Activation Function, Overfitting, Cross-Validation*

1. INTRODUCTION

There are many approaches for image classification. Few of the best ones are KNN (K- nearest neighbor), linear classification and CNN. But as the problems tend to get complex and with increasing data it is found that there is no alternative for CNN approach. CNN works by modeling small pieces of image, combining them and feeding them to the next layer of the network. CNN has three main layers they are

- Convolution layer
- Pooling layer
- Fully connected layer.

Convolution layers are responsible for detecting the edges with the help of filter to create a feature map that combines the presences of detected features from the n*n image matrix. Filter can be defined by us like line detectors but the main goal for convolution networks is that it should learn the filters during training the model with respect to the specific problem. The next layer is the pooling layer which is used for reducing the dimensions of the image. This is performed by Sampling of the input from convolution layer, with the help of filters. For example, we might have max pooling filter, where after getting the reduced matrix we can just consider the max value of the matrix. The final layer is the fully connected layer which is the final class determining step where we decide which class does the given input image belongs to. The number of neurons in the last layer is determined by the total number of categories considered accordingly with our problem statement.

The major steps to be followed before we start building our neural network model are - finding the right activation function, data pre-processing and weight initialization. Activation functions are used for regularizing the output of each node in a particular layer. There are many types of activation functions available few major ones are sigmoid, tanh and ReLU. The activation function which has been giving better results in our research is softmax function which is used for multiclass determination.

Data pre-processing is the most essential step. We should make sure that all the inputs are of the same dimensions and if required mean centring can be performed. There are many tools which come along with the python library for this purpose.

Weight Initialization, there are weights associated with each particular node of neural networks which have to be determined based on various evaluations and experiments on the networks output. The general initial approach is assigning zeros as weight to all nodes, but this is not a good suggested method to be followed. In our problem statement approach we have found out that assigning higher weight values to the categories with lesser value as compared to the classes with greater value has given better results.

Our main idea is to find out the best neural network which is most adaptable for classifying the characters in the given frame of the video. The evaluations and experiments performed resulted in determining the right hyperparameters for the neural networks which have given the maximum accuracy.

2. METHODOLOGY

(a) Handling video files in Python
(b) Reading the video to extract the frames
(c) Labeling the images for training the model
(d) Building the model
(e) Calculating the screen time

2.1 Handling video files

One of the most exciting features of python is it provides a huge list of libraries to carry out the task. Few important libraries relevant to work include:

OpenCV
numpy
pandas
matplotlib
SKimage
keras

Some of their functions are:

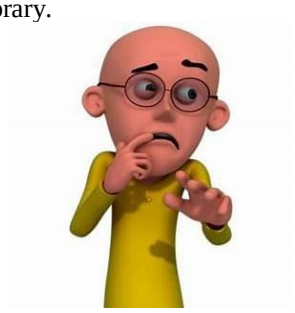
OpenCv – for capturing images
numpy – for mathematical operations
pandas - data analysis, data cleaning and data preprocessing
keras – developing the model
matplotlib – various plotting function
skimage – image processing

Not to worry!!! The point will be clear soon.

2.2 Reading the video to extract the frames

The first question here is - what is a frame?
A frame is a single, still image displayed by the computer or any other device. A video is constituted of continuous frames, which keep on changing at a very high speed, thus giving the effect of a carry-on video. Thus, the frames need to be extracted from the video to identify the characters present in the video. The frames are extracted for every second. So, if the duration of the video is 2minutes, the total number of frames will be 120.

First, capture the video via the *VideoCapture()* function , and then retrieve the frames from it and save them as images with the *inwrite()* function. These functions belong to the cv2 library.



Let's consider a video of Motu and Patlu. Identify the screen time of both Motu and Patlu in the video. In each frame, identify that which frame consist of Motu and which one consist of Patlu.

2.3 Labelling the images for training the model

After extracting the frames, figure out that which frame, consist of which character. Since a frame can contain mostly, either Tom or Jerry or both, this is a multi-class classification problem.
The problem, consist of 3 classes:

0 – Motu
1 – Patlu
2 – neither Motu nor Patlu

Example:

	Image_Id	Class
0	Frame1.jpg	2
1	Frame2.jpg	0
2	Frame3.jpg	1
3	Frame4.jpg	2

For labeling the images, make a csv file, containing each of the images names and the corresponding class to which they belong. To avoid the compiler to consider, any class superior to another, perform one hot encoding of the classes, using *to_categorical()* function which belongs to thekeras.utils library. Example, class1 or class2 should not be considered superior to class0, since they just represent presence of particular character and not any sort of superiority.

	Image_Id	Class0	Class1	Class2
0	Frame1.jpg	0	0	1
1	Frame2.jpg	1	0	0
2	Frame3.jpg	0	1	0
3	Frame4.jpg	0	0	1

Since, the images retrieved can be possibly, of different sizes, reshape them to say (224*224*3). For this, we will have to use *resize()* function of skimage.transform.

Before passing the input data to the model, perform certain preprocessing on the input, in order for our model, to give the best results. For this, use the *preprocess_input()* function which belongs to the keras.application.vgg16 library.

Validation

In order to test the model's accuracy, split out whole dataset into training set and test set, wherein, train the model on the training set and check the results on the test set. If the result obtained by applying the model, on the validation set, is same as the actual results, it can be concluded that the model will work equally fine with any video.

To do so, use the *train_test_split()* function which belongs to the sklearn.model_selection library. Here, the test_size parameter help in deciding the part of dataset to be trained on and part of it, to be used as validation set.

2.4 Building the model

A VGG16 model is being built. It is a convolutional neural architecture named after Visual Geometry Group from Oxford, who developed it. It is considered to be one of the best vision model. This step further includes 3 steps:

1. Build the model
2. Compile the model
3. Train the model

When building the model, provide the number of nodes required in each of input, hidden and output layers. The number of nodes, in the output layer, is same as the number of possible outcomes. In this case, it is three i.e. the three possible classes to which an image can belong. To compile the model use the *compile()* function. After successful compilation, fit the model, to the training data, and check the accuracy of the model using the test set. Since, convolutional neural network with back propogation, is being used, specify the number of times (epochs), the back propagation needs to be done. The number should not be very low, as it can lead to underfitting and it should not be very high as well, as it may cause overfitting.

2.5 Calculating the screen time

After building the model and training it, calculate the screen time of the characters the model is trained on, in any video. For this, capture the video, extract frames from it, reshape all the images and perform pre-processing, as above. Now the images are ready to be predicted. This can be done using the *predict_classes()* function, predict the class to which the particular frame belongs. The screen time will be stored in the array obtained using the above function.

3. OUR EXPERIENCE

3.1 What worked and what did not

In order to achieve the goals of this project, focus on every possible scenario and try to test each of them. Among all the outputs obtained, by testing each scenario, some were extremely good, while some were not that satisfactory. In this section, there are some difficulties encountered, while developing the model and how to overcome them in detail.

First issue was, that the results were not that adequate. The main reason behind this problem was, the difference between the images used in the model, which are cartoon images, and the images used to train the model which are actual images. Model could not differentiate between these images. To tackle this situation, take the following steps.

- Retrained the model using few labelled images.
- Used pre-trained model after removing the top layer.

After making above changes, the results were more adequate than the previous results.

The main feature of this project is accuracy. But the accuracy of the output was not satisfactory. For achieving the accuracy, increase the number of layers, but it might create a new problem - lack of synchronization between training and validation

accuracy. As the model was trained on particular dataset, it failed to analyse and predict unseen dataset which caused overfitting of model with low performance and accuracy. In order to tackle this situation, use combination of dropout layer and Dense layer. In this model, all the classes were not properly balanced which affected the prediction and accuracy of the model. Duration for one character was on the screen was more than the duration of some other character in the video, as a result of which, characters with high duration were given higher priority and prediction of most of the frames was determined by it. This situation affected the output accuracy, so in order to develop a well-balanced class model use sklearn.utils.class_weight module. It has a function called compute_class_weight() which assigns higher weight to the classes with lower value count as compared to the classes with higher value count which results in a well-balanced model.

Consider, each and every scenario, and test them to develop the model. It would be difficult to find which model has the highest accuracy. So, use Model Checkpointing to find the most efficient model which will produce lowest validation loss and then deploy the model to predict the output with high accuracy. This will result in a model with good sync between training and validation accuracy. In addition to exploring, try more hyperparameter tuning on a single model.

4. CONCLUSION

The vision was to develop a high-quality, well-balanced model, with high accuracy, for the prediction of the screen time of each character in a video. This model has the following accuracy rate:

Data	Accuracy
Validation data	86%
Test data	63%

Now the question is how to achieve higher accuracy? The answer is very simple, providing more training data which will increase the accuracy of the test data. The images used in this model, does not give sufficient information to the model, so, provide more images for training the model. The advice would be to create a data set by

- Extracting more frames from different videos.
- Reshaping, resizing and labelling the extracted frames according to the model.
- And last training the model with these data.

This will increase the accuracy and performance of the model. How the model is going to help and its usefulness:

- The payment of an actor depends upon his screen time in a movie or show. This model can be used to calculate the duration for which a particular actor was on screen in a show or movie.
- We can also access the information of any favorite actor in show or movie by adding the dataset of their information.

There are many other examples where this technique can be used to predict screen time of any character in any video.

5. REFERENCES

[1] Srivastava, N.: Improving neural networks with dropout. Ph.D. thesis, University of Toronto (2013).

[2] Zeiler, M.D., Fergus, R.: Stochastic pooling for regularization of deep convolutional neural networks. arXiv preprint arXiv:1301.3557 (2013).

[3] https://www.analyticsvidhya.com/blog/2018/09/deep-learning-video-classification-python/

[4] https://www.analyticsvidhya.com/blog/2016/10/tutorial-optimizing-neural-networks-using-keras-with-image-recognition-case-study/#six

[5] Ciresan, D.C., Meier, U., Masci, J., Maria Gambardella, L., Schmidhuber, J.: Flexible, high performance convolutional neural networks for image classification. In: IJCAI Proceedings-International Joint Conference on Artificial Intelligence. vol. 22, p. 1237 (2011).

[6] Ciresan, D., Meier, U., Schmidhuber, J.: Multi-column deep neural networks for image classification. In: Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on. pp. 3642–3649. IEEE (2012).

[7] Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R.: Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580 (2012).

[8] Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In:Computer Vision–ECCV 2014, pp. 818–833. Springer (2014).

[9] Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., Fei-Fei, L.: Large-scale video classification with convolutional neural networks. In: Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on. pp. 1725–1732. IEEE (2014).

