Human Action Recognition- CSC461 Project Final Report

Jamil Awada
jamil.awada@lau.edu

Hanine Al Khatib
hanine.alkhatib@lau.edu

Mohamad Dhaini
mohamad.dhaini02@lau.edu

April 3, 2023

Abstract

This paper explores a machine learning model's ability to recognize human actions based on an image dataset. Applications for this kind of technology have been on the rise, especially amidst growing security and biometrics-related concerns. Image classification is an area in image processing where the primary goal is to separate a set of images according to their visual content into one of several predefined categories. Each class is represented by a set of features (feature vector) and the algorithm that maps these feature vectors to a class uses machine learning techniques. Our dataset categorizes its data into 15 distinct classes, each representing a certain action. The labels are represented by a String object, which poses a problem that is fixed by encoding our variables into distinct mathematical values. We split the features and label into separate numpy matrices so that they can be operated on independently. We fit the data into 3 different configurations that yield different results. The ReLU activation function and having 3 hidden layers produced the best results.

1 Introduction

A surge in processing power and IoT has led to a significant increase in the usage of Image Classification techniques in computer vision. Its applications range from motion-enabled entertainment to tight security implementations in highly technologically advanced states. These applications share a common paradigm of processing data in a way that allows a machine to classify human behavior. From a shallow perspective, the task of identifying and correctly labeling human behavior may appear to be a daunting task, however previous related work proves its accessibility amongst other things. Our work seeks to understand the underlying mechanisms behind convolutional neural networks and their applications in image classification, specifically in the domain of Human Behaviour Recognition. Human actions in all their complexity and ways of being perceived can be expressed with a wide range of units pertaining to color, wavelengths emitted, motion, sound and others. These all provide unique pieces of data that should be applied depending on the scenario. A model such as ours can be applied in the aforementioned fields, as well as traffic identification, security, medical equipment, face recognition and others.

2 Related Work

It is important to note that this dataset was procured and distributed to participants of Data Sprint 76, a hackathon. This provides reassurance that the data is expected to be worked on and consequently provide certain expected results. It should be made clear that the participant's submissions and work is not in any way accessible or able to be seen by anyone else. A paper by S. Blunsden, R. B. Fisher[1] that explores multi-person human behavior detection. Datasets were scarce when the paper was conducted, and thus classes were restricted to just 6 behaviors. Performance was nonetheless impressive given the circumstances and provides a foundation for further work to come.

Another paper by Chen-YunLuoa, Shu-Yan Chenga, He Xuab, Peng Liab[2]that tackles the same problem using a streamlined EfficientNet-based model. The model proved extremely accurate, reaching levels of 99.69%. It works

by streamlining the MBConv module, adding Residual structure and selecting an optimal activation function. Data is collected using cloud data LiDAR technology. Collected points are transformed into images to be used as the dataset. The improved EfficientNet model is trained using this dataset, and it is also used with other classification networks such as DenseNet, GoogLeNet, and MobileNetV3.

Additionally, a paper by Zehua Sun, Qiuhong Ke, Hossein Rahmani, Mohammed Bennamoun, Gang Wang, Jun Liu[3] that presents a comprehensive survey of recent progress in deep learning methods for human action recognition based on the type of input data modality.

3 Proposed Method

A traditional neural network is comprised of the standard input, hidden and output layers. In essence, the input layer receives a wide range of diverse inputs, the hidden layers calculate using the inputs, and then the output layer displays the results of the computations. Each neuron in a typical neural network has a unique weight, and they are linked to other neurons in the layer below. This implies that the data being entered into the network is not subject to any assumptions, a boon for image classification models such as ours.

Our model is based on convolutional neural networks, a type of deep learning model used for processing data in a grid pattern, which is ideal for our image classification application.

Convolutions serve as the foundation of CNNs. The neuron in the next layer only receives pixels that are covered by a convolutional kernel. The local spatial associations present in a picture are therefore captured by CNNs. For applications involving picture data, CNNs allow for a reduced number of weights, making the models lighter and simpler to train while yet having the same or greater prediction ability than fully linked networks. Operations performed in each layer are could be linear transformations or non-linear transformations. The non-linear transformations are the activation functions. There are many of the latter that are widely used in CNN, of which we will be using are Softmax, Sigmoid, and ReLU activation functions in our experiments.

We employed the Adam optimization approach, a stochastic gradient descent modification that has lately gained more popularity for deep learning applications in computer vision and natural language processing.

We used Categorical Cross entropy loss, likewise known as Softmax Loss. It is a Cross-Entropy loss combined with a Softmax activation. We can train a CNN to output a probability across the 15 classes for each picture if we utilize this loss.

4 Data set

The data set we have opted for contains over 12000 labelled images split among 15 classes of human activities. Each image can only be labelled one class. The 'Train' file contains all the images used for training, with each folder containing all the respective images. The 'Test' file contains around 5400 images of human activities, all unlabeled. They will be used to test our model on. The file also contains 2 .csv files, which represent the images in the same order they are stored in. It is thus very important that the data not be reordered, or at least accounted for.

With the simple nature of the data and the dataframe consisting of only 1 feature, exploring it is straightforward. The images are clearly stock photos that do not require much context for someone to understand them. A simple plot reveals that the data is evenly distributed among the 15 classes, each having 840 rows.

Originally, the thought of resizing images to a standard did not seem like a good idea because stretching or compressing an image on the X or Y axis could lead to blurriness or uneven proportions of the elements within. However, after some research, resizing the images to a common value was encouraged to promote consistency within the model. Plotting the pictures reveals that some of them were almost twice as wide as they are tall, and vice versa. Below are figures illustrating the images before resizing as shown in Figure 1 and Figure 2.

Figure 1: Example 1 illustrating unprocessed images and their sizes.

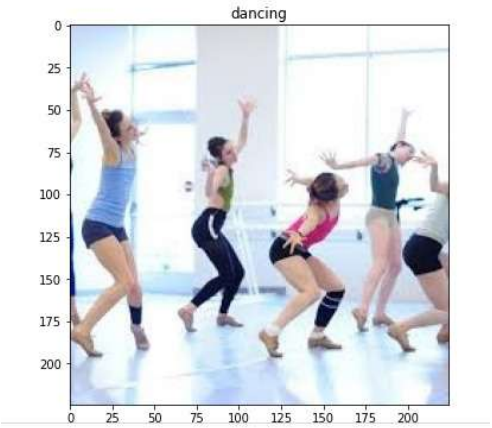

Figure 2: Example 2 illustrating unprocessed images and their sizes.

4.1 Preprocessing

Resizing images shouldn't pose a problem as most pictures have the action being depicted in it in center-frame. This works well with the fact that after resizing, larger images are given black borders and smaller ones are not stretched too much rendering them misinterpreted. Another positive to this resizing is that it avoids overtraining on some specific image orientation or size. This could backfire as the model could be overtrained on clearer pictures and perform worse on blurrier ones.

Without further examination, it's unclear whether this image data needs to be cleaned. Possible issues that could arise would be significantly different picture sizes, grouped similar RGB values (colors blending too easily), unclear or vague behaviors, multiple people in the same shot, etc.. . A deeper look into the dataset reveals that the images are clear, with distinct colors and actions being adequately represented. Minimal image preprocessing needs to be done.

A possible problem that could arise is the presence of images of people doing multiple things at once. For example, a person could be both sitting and listening to music at the same time. In cases such as these, how would the model label it? This is to be explored further down the road. On the other hand, our label in the dataset is a categorical value. This presents a problem as our model cannot interpret text while creating the model. A solution to this would be to encode it using LabelEncoder. Each category of action is thus now represented by either a 0 or 1 indicating whether the action belongs to it. Below are figures illustrating the images after resizing as shown in Figure 3 and Figure 4.

Figure 3: Example 1 illustrating processed images to their new size.

Figure 4: Example 2 illustrating processed images to their new size.

5 Experiments

While fitting our data, we experimented with different parameters the TensorFlow code takes in attempts to both reduce run time and increase accuracy. While they were mostly unsuccessful, they did provide insight into how the parameters influence the model.

Experiment 1: Increase the number of strides taken to reduce runtime.

We were surprised at first to know that the code takes as long as it does. This would be our first time fitting a deep learning model and figured we had done something extremely wrong for the code to take well over a few minutes to run. Some digging showed us that the runtime was completely normal. Before knowing this, we experimented with increasing the number of strides taken from 2 to 8 at each layer. We had intuitively figured it would signify the magnitude of the step taken, but frankly did not understand its mechanism in context.

Result 1: Runtime was significantly decreased, but at the cost of harshly reduced accuracy.

From our understanding, the strides parameter indicates how many steps the convolution will take at a time before reaching a border. Re-examining the experiment with this knowledge in mind explains the results we got.

Experiment 2: Decrease the kernel size to increase accuracy.

Kernel size refers to the number of pixels in both height and width the convolution window takes at a time. Logically, we figured that processing a fewer number of pixels at a time would lead to more resources being allocated to them instead of being shared amongst a larger number of them. We decided to run the model with this in mind, knowing that our deduction is most probably wrong but still curious.

Result 2: Runtime was too long and the program had to be aborted.

Each epoch took well over 6 mins, and with the number of epochs we had it meant that running the program would span overnight and leave us with no time to readjust.

Experiment 3: Increase the number of epochs beyond the original threshold.

Our initial run had just 20 epochs, thinking that was already plenty given the industry standard. While we did start noticing marginal returns beyond that point, we did decide to keep going and see how the model would alter the more we add. As expected, it began giving us diminishing and even negative returns.

Results 3: Accuracy increased.

Surprisingly, accuracy started going up noticeably at the 40th epoch mark. There was a 5% increase from the original threshold, and a 2% increase at the 60th epoch from the 40th. This nets us an additional 7% from the original threshold, an impressive number that we would not have had if we stayed in fear of overfitting.

6 Configurations

Our model was configured 3 times. One variable was altered at a time while changing configuration, in order to single out the determining factor to better accuracy. We focused on altering the number of hidden layers and what activation function was being used, as to not make changes unnecessarily overly complex and hard to keep track of.

The first configuration is comprised of 2 hidden layers, each having Softmax as the activation function. The output layer has Sigmoid as the activation function.

The second configuration increases the number of layers from 2 to 3, and keeps Softmax as the activation function. The output layer keeps Sigmoid as the activation function.

The third configuration keeps the number of layers at 3, but opts to replace Softmax as the activation function with ReLU. The output layer remains unchanged as well.

7 Results and Discussion

7.1 Results

Results show high discrepancies between all 3 configurations. Our best performer was configuration 3, which scored an impressive 98% accuracy at epoch 60, while configuration 1 sits at the bottom with a lackluster 74% accuracy at epoch 60. Configuration 2 sits in between with a decent 89% as shown in table 7.1 and Figure 5. As previously mentioned, some experiments that could have yielded better or worse results took too long to run and didn't feel worthwhile to explore with the result we got.

Figure 5: Figure illustrating the resulting accuracy after configuring model for the third time

Configuration	Accuracy
Config 1	74%
Config 2	89%
Config 3	98%

Table 1: Accuracy of Configurations

7.2 Discussion

Initially, we built the model under the assumption that Softmax is the go-to activation function used in the hidden layers. Similar to the logistic function, Softmax gives us a categorical output. The result, however, is multinomial. In the standard digit recognition scenario for example, Softmax provides a categorization of the digits 0 to 9. Our first run with configuration 1 had just 2 layers and Softmax as the activation function. We pinned its bad performance on the low number of layers instead of examining the activation function, thinking that it makes sense for an image classification application to use Softmax on the hidden layers. It didn't take us until after configuration 2 ran for us to question its viability as the activation function of choice. We then ran the same configuration but with ReLU as the activation function instead, which yielded much better results as seen in the table above. Research was conducted and we concluded that Softmax was supposed to be used as the activation function on the output layer, and not where it was being applied. While we figured we would run a hypothetical fourth configuration for better results, the accuracy achieved by configuration 3 was more than enough for us to be satisfied. The number achieved was already very high and tipping on the edge of overfitting (if not being somewhat overfitted), and any further increase would be marginal at best or detrimental at worst. We were left puzzled and questioned the status quo placed on the activation functions in terms of which one fits where, but these questions allowed us to further explore their uses. Our conclusion was that the model with layers having the ReLU activation function converged faster than its Softmax counterparts, which could indicate overfitting. Interesting to note is that each epoch took the same amount of time despite differences in activation layers, however this can be explained by the change in other variables.

The number of layers fluctuating from 2 to 3 hidden

layers is in line with what we expected. Models handling less complex and simpler data usually have 1 to 2 hidden layers (such as the MNIST), while 3 to 5 hidden layers are used for more complex applications. A higher number of layers would expose us to a greater chance of overfitting.

8 Conclusions

Our model showcases a taste of what computers can interpret from what they see. It shows us the intricacies and potential in the field of computer vision and how a simple application of a machine learning program opens the doors for opportunity and creativity. From a technical perspective, the different configurations and the process as a whole highlight the importance of analysis and a thorough understanding of the model used. It is not enough to procure a dataset and run any model on it without examining the nature of its content . Preprocessing, model assessment, parameter tweaking, encompass some of the many steps needed to create an accurate realistic model that can be used to correctly predict new input.

Further research in the field should be (and is) being done to stretch its capabilities to its limits. Our application covers images. Our application covers images, though its real-world use covers other formats such as video and audio (by listening to the sound of footsteps being taken, for example). Interesting to note the recent surge in applications of Computer Vision such as DALL-E 2 and such models gaining popularity. The data science field is constantly being shook by new innovations and breakthroughs, and who knows where we see ourselves later.

9 Software

We used Google Colab for its cloud based computing capability. Moreover, Pandas was used to properly display and manipulate DataFrames. MatPlotLib was used to plot relevant data. TensorFlow was used to build our model. SkLearn was used to build our categorical model.

10 Contributions

We opted to work simultaneously on each step of the way instead of dividing tasks amongst ourselves. The long running time of the algorithms meant that we had enough to think of strategies and further steps to take. Meetings were held in person and online. The online meetings were held when checking up on an algorithm's runtime while analysis and design were done in person. The paper was equally divided and each member had to write their own share of the sections, although we did look over each other's work and offered input and corrected any mistakes. The notebook was shared to everyone, however Mohammad's internet speed allowed him to fully upload the large dataset onto Google Drive, so most of the work was done on his laptop. The physical aspect of writing code pales in comparison to the thought process that went behind designing the model, as TensorFlow provides us with most utilities we needed in premade functions making most of our work a collaborative effort to think of the best possible configurations given what we know.

References

[1] S. Blunsden and R. Fisher. The behave video dataset: ground truthed video for multi-person behavior classification. *Annals of the BMVA*, 4(1-12):4, 2010.

[2] C.-Y. Luo, S.-Y. Cheng, H. Xu, and P. Li. Human behavior recognition model based on improved efficientnet. *Procedia Computer Science*, 199:369–376, 2022.

[3] Z. Sun, Q. Ke, H. Rahmani, M. Bennamoun, G. Wang, and J. Liu. Human action recognition from various data modalities: A review. *IEEE transactions on pattern analysis and machine intelligence*, 2022.