Computer Vision project: Dance Classification

Daniil Smoliakov
Daniil.smoliakov@polytechnique.edu

Guillaume Loranchet
Guillaume.loranchet@polytechnique.edu

Abstract

Understanding and interpreting the movement of people is essential for various application.Especially with the increase in the amount of data (video and images). And the active development of video cards capable of processing large amounts of information in parallel. This report offers a solution to the problem of classifying the type of Indian dance based on one image from it. The main goal is to learn how to use the person's appearance and posture.

1. Introduction

1.1. Instruments

For this project we use Python programming language, PyTorch as neural network programming framework, OpenCV2, PIL, Numpy and Pandas as data handling libraries. MediaPipe was choosen as instrument for pose estimation. All work was done in Google Colab to have one repository of code.

1.2. Data processing

The dataset consists of 364 images belonging to 8 categories, namely Manipuri, Bharatanatyam, Odissi, Kathakali, Kathak, Sattriya, Kuchipudi, and Mohiniyattam.

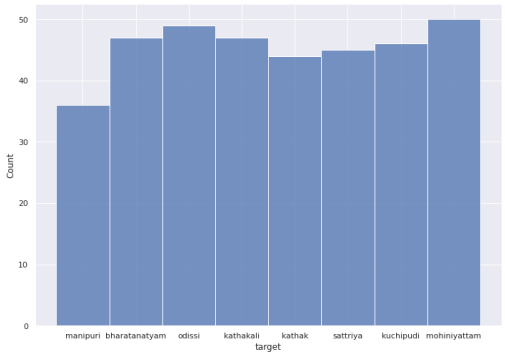


Figure 1: Histogram of different dance type distribution

As can be seen from histogram distribution of labels somewhat uniform. Overall we do not obtain a lot of data and for each class we have around 40 images with examples.

Other important aspect of data is ratio of images. Almost every image has unique ratio of size. To handle them equally and preserve as much information as we can we used two approaches: just resize, not preserving ratio of image and padding with following resize that will preserve ration.

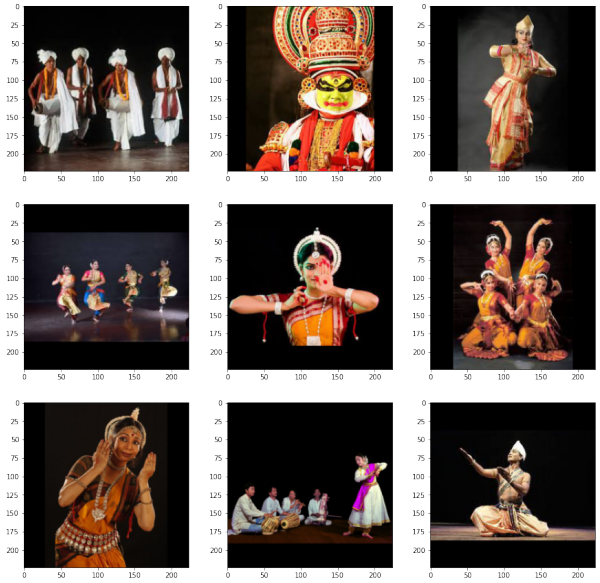


Figure 2: Examples of pictures present in the dataset (with padding)

2. Classification

For the first and baseline solution of the classification problem, we decided to optimize the operation of resnet18 specifically for our task. We will take advantage of the fact that in the process of learning on the image, the resnet learns to detect features, which we can then use correctly to solve our specific problem. This process is called fine tuning and is part of the transfer learning.

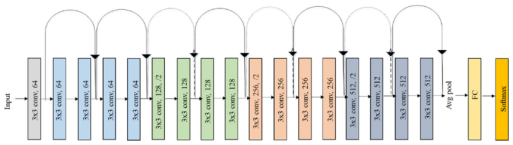


Figure 3: Original ResNet18 architecture [3]

Even the smallest resnet architectures are quite big and training training them again would take a long time. Instead, we will replace the last dense layer to fit our ouput size and lock the gradient in all gradients except for the last few layers. This will train more the last layers and provide us a better baseline.

To be able to properly work with data in PyTorch we created our own dataset class - **BaseDanceDataset**. This class provides an interface to iterate through the entire data while transforming the data in a form understood by PyTorch dataloaders. Additionally we implemented padding transform to add a black padding instead of stretching the images.

We randomly split our data in train and validation set with proportion of 0.7 and 0.3. However we are fixing splitting seed, so we will obtain similar results after split, that is important for work of early and late fusion networks.

For optimization we are using PyTorch implementation of Adam algorithm with learning rate equals to 0.0001. To have a better control over learning process, we shuffle the data and work with batches of size 32. After 4 iteration of train function we obtain an accuracy of 74.5% for stretch images (67.3% for padded images) on the evaluation dataset and 100% on training.

Figure 4: Loss function value on train and evaluation data

From the loss plot we can see how loss values change on train and evaluation set. Any further expansion of learning process do not give any better results. Even more, we can see that we are already overfitting a bit for only 4 epoches.

2.1. Pose Estimation

The next step is to use a human pose as a guide for algorithm. Since there are people in all the images, we can try to locate the arms, legs, face and other important points of the human body. We can then send this data directly to the neural network, hoping that it will be able to identify these features and use them in the process of classification and training.

To do this we used the MediaPipe library. For pose estimation, they utilize two-step detector-tracker ML pipeline. Using a detector, this pipeline first locates the pose region-of-interest (ROI) within the frame. The tracker subsequently predicts all 33 pose keypoints from this ROI. [1]

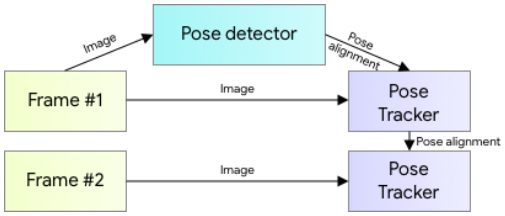


Figure 5: ML framework

However we are only utilizing the static image part of framework.

To give more sense of the results, we provide one annotated image here [6].

Figure 6: Estimated pose

As can be seen, MediaPipe is correctly setting landmarks based on BlazePose model, even if we cannot see the whole body.

To work with annotated images, we have to modify the dataset class. First part - precomputing: we annotate ev-

ery image in our dataset, so we don't need to annotate in on-demand mode. They are then stored in an array with annotations. To be able to use PyTorch dataloaders we creat an additional Dataset class, that will store not only the paths and names of the images, but also the array with annotated images, to be able to access them on request.

Otherwise, the overall procedure stays identical: same learning process and same evaluation methods.

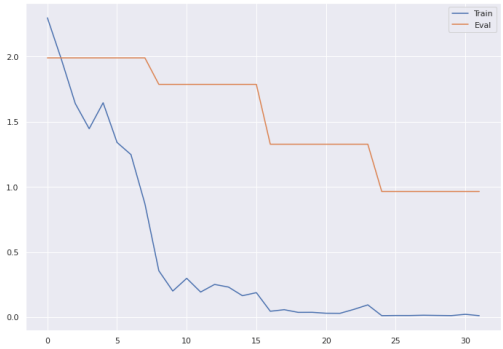


Figure 7: Loss function value on train and evaluation data with pose estimated images

Overall we are again performing just 4 iterations of learning and getting a final accuracy equals to 76.4% which is bit better than in the previous method. There are multiple reasons for this, maybe the network managed to efficiently use the skeleton drawn above the picture for the classification. It is always important to remember that we are working with a quite small dataset and even one image can make huge difference on score. Indeed, we sometimes obtained a worst accuracy with pose estimation with the exact same configuration.

2.2. Early and late fusion

The last part of the solution is to try to use early and late fusion to improve the classification results.

2.2.1 Late fusion

At the moment, we tried to get two different approaches to solve the same problem: classification based on images and classification based on pose estimation. It seems natural to try to capitalize on the two created solutions by using their results at the same time. For late fusion, we'll just use the weighted sum of our two different approaches. At the same time, it is desirable to obtain results that will be at least somewhat better than those that we have in both algorithms.

As can be seen on image we separately handling data and algorithm, retrieving result from each CNN and then mix them. In our case it is just a weighted sum with both weights equal to 0.5

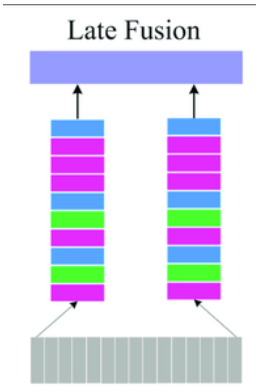


Figure 8: Late fusion schema [2]

We are scoring 78.2% of accuracy on evaluation set, which is very similar to the one obtain with pose only.

2.2.2 Early fusion

Things get more complicated in relation to early fusion. On paper, you just need to merge the data and fine-tune the network architectures so that it can receive and process the new data format.

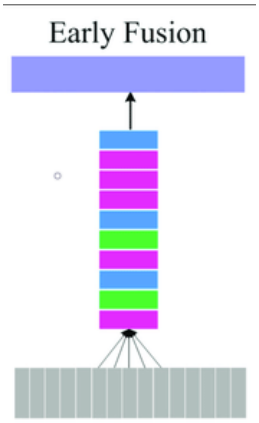


Figure 9: Late fusion schema [2]

Which raises new questions. How to set weights for three new dimensions in our case. One of the obvious options: random weights and a copy of the original ones. The first option gives small results after training, since it requires training those very new weights. But our small dataset does not allow enough training. The second option is simpler and gives much more acceptable results.

However, the question still remains where to mix the data. In this work, we consider two options: the first is simply data fusion at the input level and data fusion before all

the dense layers of the network, by creating new additional network consisting only of dense layers.

Both are early fusions approach, but the first one is much easier to implement, so we will start with that.

To implement the first approach, we will write another class that will work with our dataset. At the moment, we want to receive, on request, not just an image, but simultaneously a merged clean image and an image with annotation. As a result, we get an image of size $224 \times 224 \times 6$. For more optimal work, we pre-calculate annotations and simply pass them to the class of this dataset.

Next, we need to modify the resnet architecture so that it is capable of accepting data of this given format. We thus replace conv1 with a new layer, which accepts not only 3 but 6 channels. At the same time, we separately save the existing weights to later assign them to both parts of the new layer.

The rest of the learning process remains exactly the same: unlock several layers and do fine tuning for our task.

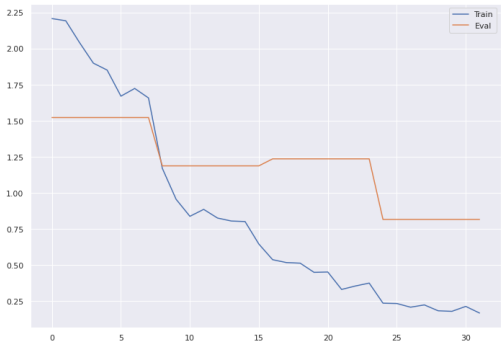


Figure 10: Loss values with early fusion approach [2]

As result we achieve accuracy of 72.4% on evaluation set.

The second approach will require the creation of a model, not a dataset. It will be based on a pretrained resnet, but we will remove the last dense layers. And we will create an additional layer, which will be twice as large as that of the original reset, this is 1024 elements for input. We will use the dataset from the previous approach. Also we are changing forward pass of neural network. We split the image back into two parts: clean and annotated. Then we pass both parts through the same network (resnet without the last layer), get the features after the convolutions, combine them into one tensor with a length of 1024 and simply run them into an additionally created layer, getting a prediction at the output.

Otherwise we preserve exactly the same procedure for training and evaluating.

Final result for this type of accuracy is 71.5% of accuracy.

3. Class activation maps

Class activation maps is the process of highlighting some parts of an image depending on their importance on the final classification. It gives more insight into the learning of the network and allows us to see what is the model looking at in the image. It could be use as a debugging method or to check that the model is correctly learning. To obtain this heat map, we feed an image to the network and select the weights of the final layer responsible for the given classification. Then, we multiply each feature map of the last convoluted layer with its respective weight and sum them to obtain a single image. We finally use up sampling to extend the size to the one of the input image.

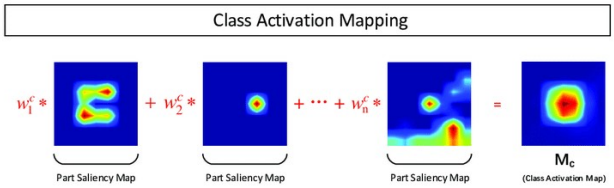


Figure 11: Structure of Class activation maps

Class activation can be applied to any layer we want. If we have a look to early layers,we would have smaller and more areas highlighted (see figure 12) as a CNN first identifies the details in the image and then increases the range to finally focus on a single area of interest. To do that, we only have to select the weights from the previous layer and multiply them with the one of the layers after to obtain the right dimensions. We can then apply the same method as before to get an activation map. We use the official implementation of the Class Activation Paper [4]

3.1. Interpretation and limitations

The main limitation we had to face was the size of the dataset. Indeed we only had 520 images, which makes an average of 65 images per label that then needs to be split into a training and a testing set. We think that due to this limitation, our model was overfitting and only memorizing information. This could explained the small interpretability of the class activation maps. Indeed, if the model is not actually learning, it is normal that the highlighted features will not be the most relevant.

Another limitation is the composition of the images. Indeed, we can notice on 13 that in most cases, the most relevant information for the model are situated on the boundaries of the image rather than on the person itself. Our interpretation is that it is easier for the model to focus on the background to predict a label. If true, it is problematic as the model is supposed to learn to classify dances and thus focus on the people dancing, their clothes... To remediate

this problem, we should have the same uniform background on all images to avoid bias.

On figure 12, each column represents one label with each time two examples of the class activation map plus the image it corresponds to. We can see that for each identical class, the map are similar, as the weights outputting a given class must be similar. However, it is the further we can go in term of interpretation for the layer3 as each map does not seem particularly relevant to a given class.

4. Conclusion

We have implemented various approaches to solve the problem of dance classification based on photography. From ordinary fine tuning to costly early fusion.

	Accuracy padded	Stretch
Clean image	68.2%	74.5
Annotated image	67.3%	76.35
Late Fusion	72.7%	78.2
Early-early fusion	66.4%	72.4
Early-late fusion	69.1%	71.4

Table 1: Accuracy comparison

As a result, the least costly methods show a result that is not very different from the rest. And late merge is superior to all other methods. In our case, a lot is tied to the data that we have. There are about 40 images in each of the classes, and this is not enough to use early fusion in almost any form. However, later, based on fine tuning of two models using complementary data, it brought a positive result on the evaluation dataset.

4.1. References

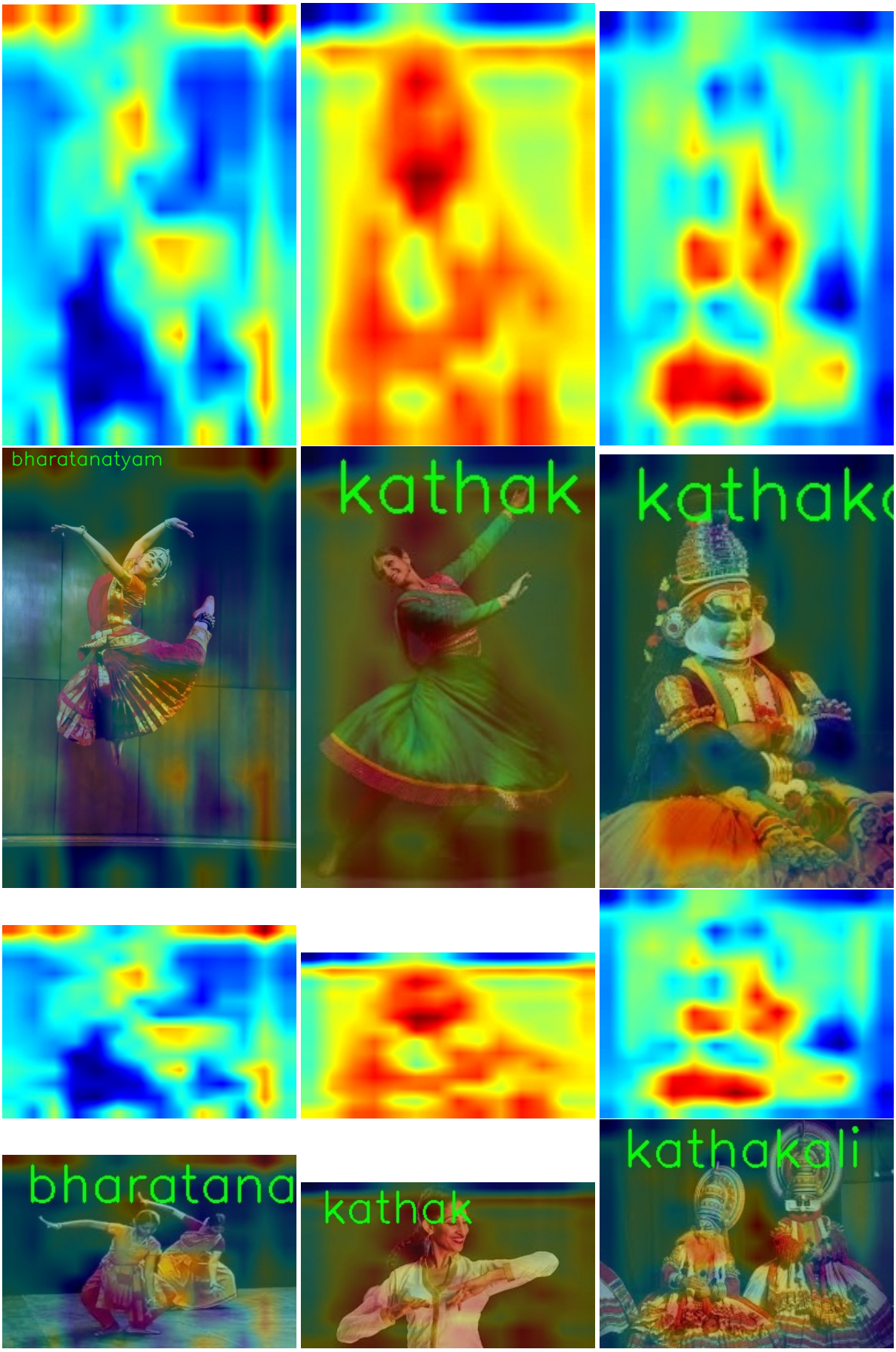
References

[1] Valentin Bazarevsky et al. “BlazePose: On-device Real-time Body Pose tracking”. In: *ArXiv* abs/2006.10204 (2020).

[2] Min Peng et al. “Dual Temporal Scale Convolutional Neural Network for Micro-Expression Recognition”. In: *Frontiers in Psychology* 8 (2017).

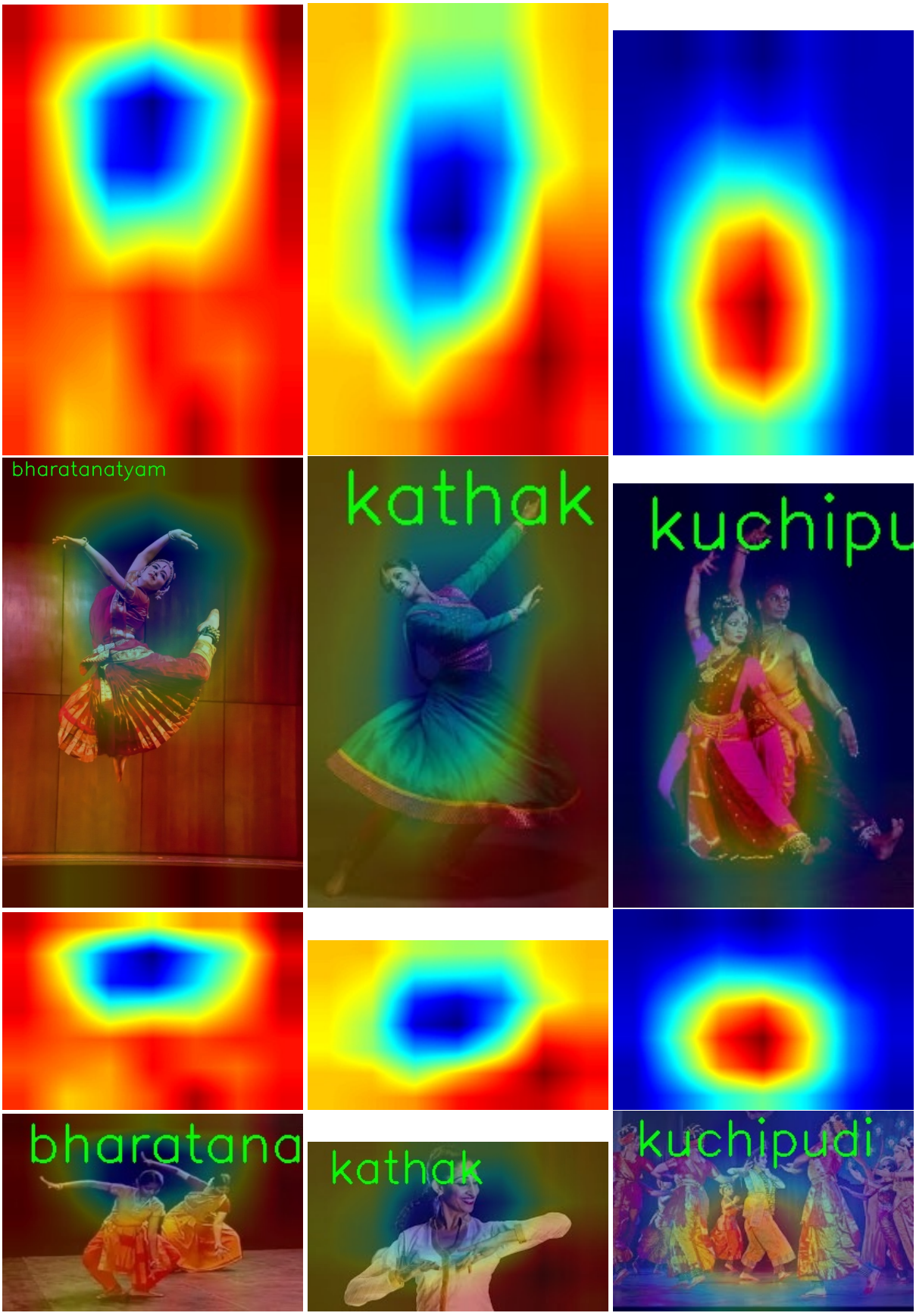
[3] Farheen Ramzan et al. “A Deep Learning Approach for Automated Diagnosis and Multi-Class Classification of Alzheimer’s Disease Stages Using Resting-State fMRI and Residual Neural Networks”. In: *Journal of Medical Systems* 44 (2019).

[4] B. Zhou et al. “Learning Deep Features for Discriminative Localization.” In: *CVPR* (2016).



(a) Bharatanatya (b) Kathak (c) Kathakali

Figure 12: Class Activation for layer3



(a) Bharatanatyam (b) Kathak (c) Kuchipudi

Figure 13: Class Activation for layer4