COL-341 ASSIGNMENT-3 REPORT

Yoga-Pose Detection

BY

KURISETI RAVI SRI TEJA-2019CS10369

SHASHWAT SAXENA-2019EE10524

Contents

1 Introduction

Human activity recognition is one of important problems in computer vision. It includes accurate identification of activity being performed by a person or group of people. In this assignment, we will be classifying yoga poses into different classes(Asanas). We have around 19 types of Asanas in our dataset and 29K images shot in variable angles for training a machine learning model. Yoga pose estimation has multiple applications such as in creating a mobile application for yoga trainer.

2 Experimentation Details

2.1 Pre-Processing of the Data

Since the data contains images taken of different people in various angles,we need to pre-process the images carefully to ensure that we do not lose any useful information.For this we implemented various types of transforms provided by the **torchvision** library.This included rotating a small fraction of images in training set by small angles(upto 20°) to ensure that the model can capture maximum possible orientations. We also implemented a random horizontal flip with probability of 0.5, Color-Jitter to ensure that there are images from different brightness and contrasts to avoid over-fitting lighting.We also normalized the images to ensure that the all values of tensor lie in range [0,1] and ensures convergence of the model.

2.2 Training Models

Since the training data consisted of images,we preferred not to use neural networks with only linear layers as they have very low accuracy with such training data.Hence we began the experimentation with convolutional neural networks and tried with different types of architectures.Initially we began with 18 layer deep CNN that consisted of 4 convolutional layers, 4 ReLu activation layers,4 Max Pooling layers,4 Batch Normalization Layers,a fully connected linear layer and a drop out layer which gave a meagre accuracy of 25% on the

public test-dataset.

Hence we thought of increasing the depth of neural network and for this we chose pre-trained models in **pytorch** such as ResNet,DenseNet,Efficient-Net as they were known to provide higher accuracies for most of the large public datasets such as MNIST,CIFAR-10 and CIFAR-100.We began with the ResNet architecture but could get only less accuracy percentage (close to 25%).So we thought of using DenseNet architectures which were relatively larger than ResNet and as expected they gave an accuracy close to 65% but with lone disadvantage of unable to use the entire training image due to the larger model size.

After shifting from DenseNet to Efficient-Net architectures,there was a large raise in accuracy percentage i.e., with all the remaining features fixed the test accuracy percentage became 75%.So we tried with different architectures in Efficient-Net i.e., B1,B2,B4 architectures.But due to different sizes of the model architectures,we had to resize the image accordingly to the model size.Out of all such permutations,we got the best accuracy with B4 architecture with image of size 220x220 and it didn't improve much with increase of image size from there on.We also tried with smaller version of efficient-net V2,but it was almost as accurate as B4.(There were larger version of efficient-net with more parameters,but they were too large that we couldn't accommodate sufficiently large image in it which lead to lower accuracies).

2.3 Parameters Used

We used the Cross-Entropy loss as the training loss owing to the fact that the pose detection was a classification problem.For gradient descent,we used Adam optimizer and added a weight decay to avoid over-fitting by regularization of weights.For training our model we only used 8 epochs to avoid over-fitting of data.

3 Conclusion

We only experimented with various types of CNN's which gave an accuracy close to 80% on test data-set.The test accuracy can be made much better if we could use open source pose detection models such as Media-Pose(by Google),Open-Pose(by CMU) which we could not accommodate in our model due to lack of proper GPU support for these open-source models.(Each image took around 2-3 seconds for detecting key points on Colab CPU and it would take around 60000 seconds=17 hours roughly for entire task and is not feasible in our case.)

4 External Codes Used

1. https://github.com/lukemelas/EfficientNet-PyTorch
2. https://www.kaggle.com/nroman/melanoma-pytorch-starter-efficientnet
3. https://www.kaggle.com/pinocookie/pytorch-dataset-and-dataloader