COSC 490 Project Report

Edouard Eltherington, Veronica Jack, Khai Luong and Logan Parker

COSC 490: Student-Directed Seminar

Dr. Yves Lucet

April 11, 2022

Abstract

Due to advances in technology, as well as circumstances surrounding the Covid-19 Pandemic, demand for at-home workouts has reached an all time high. In order to improve the accessibility of these types of activities, we have implemented a yoga pose classification platform to ensure users can perform yoga poses correctly. This platform is not meant to replace a certified Yoga instructor, but we hope it can be used by individuals performing home workouts to learn poses and practice with a model's confidence as feedback. A simple and common approach for this task is to use a single image classification model, such as a Convolutional Neural Network (CNN), to classify a user's pose. During initial stages of development a simple CNN was implemented; however, it was overfitting and did not perform well when classifying unseen data. In order to circumvent this limitation, our task was then divided into two main objectives: pose detection and pose classification. For pose detection we use the OpenPose framework to detect the user's pose, which is then fed into an image classification model, to perform pose classification. In earlier stages of the application, we developed a simple CNN which was later outperformed by the pretrained EN model. The combination of the open pose detection framework and the pretrained EN model was developed into a Yoga Pose application that provides accurate pose feedback to users.

Introduction

Since 2020, COVID-19 resulted in a rise in popularity of at-home exercising due to public health measures including the closure of gyms and social distancing (Lesser & Nienhuis, 2020). Yoga is an exercise that is welcoming for all ages and provides health benefits such as improved health, strength, flexibility, fitness, and a reduction in depressive symptoms (Shaw & Kaytaz, 2021 & Jain et al., 2021). According to Andersson and Andreasson (2021), over the

course of the pandemic, group fitness instructors have had a negative attitude toward streaming classes and most were reluctant to teach online. In a virtual setting, instructors encounter many challenges, such as providing motivation and appropriate feedback to their clients for encouragement and to prevent injury. Learning Yoga and avoiding injury becomes more challenging when an individual attempts poses on their own without the presence of an instructor. Our goal was to provide a tool for clients, and potentially instructors, to address some of these issues. In order to create such a tool, we trained a Convolutional Neural Network (CNN) and an EfficientNet (EN) model to learn common Yoga poses and recognize when a user is correctly and incorrectly performing certain poses. To accomplish our goal, we created an application that uses a pose detection framework, along with a pre-trained EfficientNet model from tensorFlow that accurately detects and classifies 14 common yoga pose positions.

Datasets

The EfficientNet model used within our application was trained on a modified version of the publicly available “Yoga-82” (M. Verma, et. al) dataset. Modifications done to this dataset included: translating the yoga class names from Sanskrit to English, hand selecting poses which were both common and contained enough data to provide us with a reasonably sized dataset, and transforming the images into a skeletal reconstruction of the original via OpenPose. These changes were necessary, as processing via OpenPose not only reduced bias within the data, but also assisted in improving the accuracy of the EfficientNet model. We suspect that these improvements are a direct result of the reduction of noise within the training and testing images.

Figure 1

OpenPose detection for Goddess

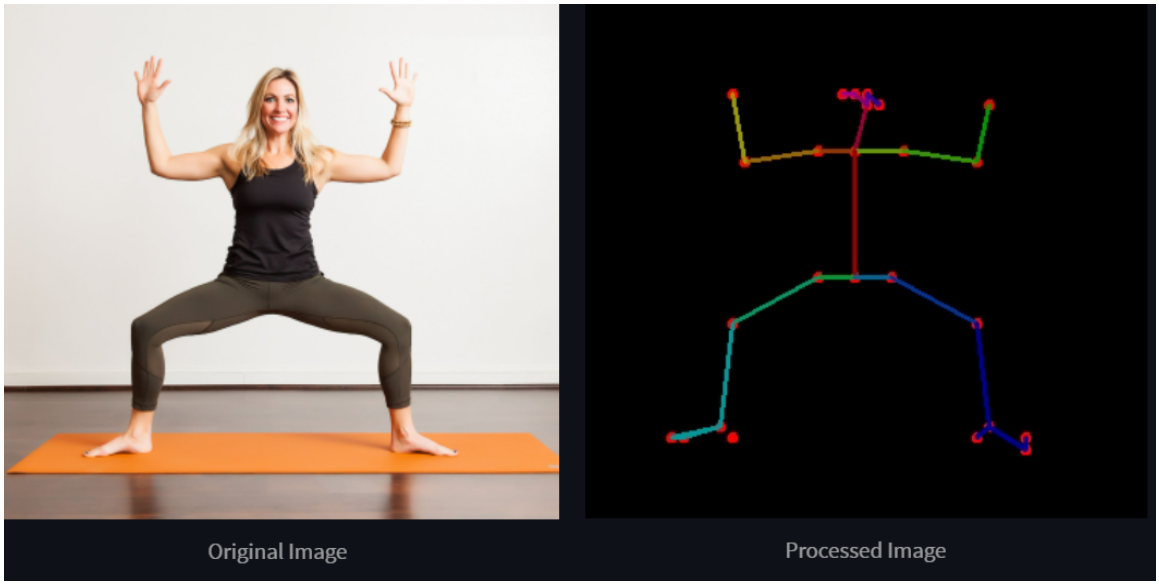
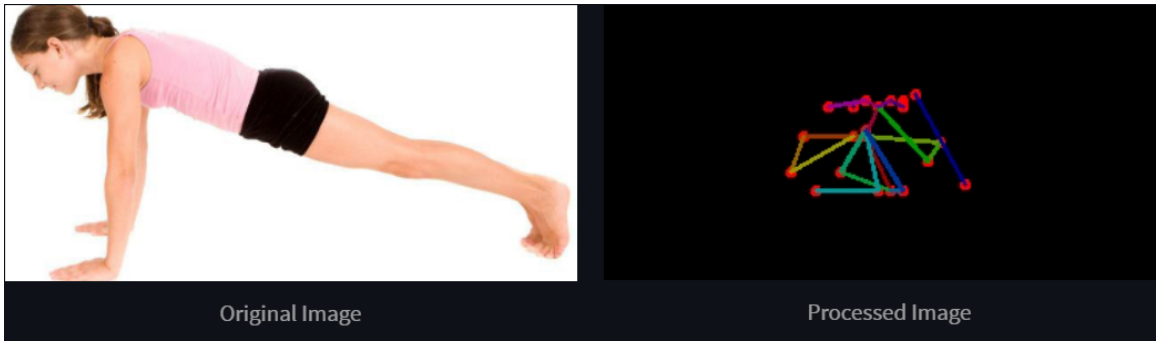


Figure 2

OpenPose detection for Plank



Although performance was increased, the processing of images using OpenPose did introduce difficulties with recognition of certain poses. Yoga poses which may be considered common, such as the plank pose, were not easily recognized by OpenPose due to overlapping limbs (Figure 1), and as a result these images are unrecognizable for the image classification model. In order to handle this issue, the unrecognized classes were removed from the dataset. After processing the data, our modified dataset consisted of 1,197 images belonging to 14

classes. When training the EfficientNet model, the training set consisted of 962 images and the validation set consisted of 235 images.

Models

For the classification task, the finalized application uses a pre-trained EfficientNet model; however, other models were trained and tested before the final model was chosen for the application.

CNN Model

The first model was a customized CNN which consisted of: one data augmentation layer, three convolutional layers, three pooling layers, one dense layer, and a classification layer (Figure 3). Before training, images were fed into an image generator which scaled the images down to a 180 x 180 pixel image. Then, 80% of images were placed in a training dataset, and the remaining 20% were placed in a validation dataset. During the training process, images were fed through with a batch size of 32. Each image was initially passed through the CNN, then through a dense layer for classification.

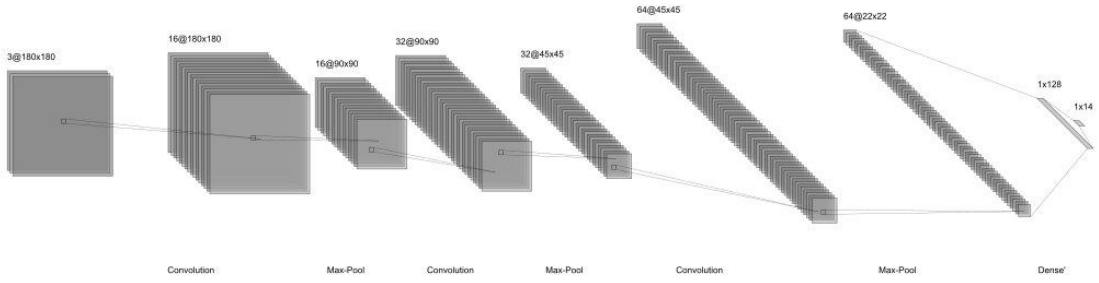
This initial CNN model was modified frequently throughout the course of the project, with attempts being made to reduce overfitting and improve performance. Improvements made to scale up the performance of the CNN included adding a data augmentation layer, improving the training dataset, increasing the number of training epochs, and changing the number of layers in the model.

The first iterations of the model had very poor training and validation accuracies of approximately 20%. By experimenting with the number of layers in the CNN, training and validation accuracy increased to approximately 35% and 20%, respectively. Adding a data augmentation layer into the CNN slightly reduced the overfitting issue, but it did not improve the

model's performance accuracy. Another method of improvement was increasing the number of epochs from 5 to 15. The training accuracy increased substantially to over 80%. Despite this, the model was still overfitting on training data, with validation accuracy remaining low at 50%.

Figure 3

CNN Model Architecture



After making these modifications to our custom CNN, we reached a point where adjusting the model parameters in TensorFlow was not enough to increase the performance of the model. We determined that we needed to have a model that could perform human pose detection rather than object recognition. In other words, we needed our model to recognize a yoga pose, then run that pose through the CNN to match it with one of the 14 classes.

To accomplish this human pose detection task, we used OpenPose, which is a system that jointly detects human body, hand, facial, and foot key points on single images (Hidalgo et al., 2022). Rather than training our CNN model on raw images, we ran our dataset through OpenPose to create a new dataset of rendered images so that our CNN model would be trained on human pose images (Figure 1). This dataset modification showed significant promise, with our CNN outputting a training accuracy of ~85% and validation accuracy of ~75%. Despite this, the CNN model was still overfitting, with accuracies not satisfactory for our pose classification

objective. This was the final version of our custom CNN. This model was saved and used as a benchmark version of the application. To further improve performance and reduce overfitting, we decided to implement a pre-trained model.

EfficientNet Model

In order to achieve our desired performance, we incorporated a Keras image classification model that was pre-trained on ImageNet. This model captures generalized image features, and can be reused in a variety of image classification applications (TensorFlow, 2019). In order to achieve higher performance on this model, the top layers were unfrozen and retrained, and a new classification layer was incorporated.

The pre-trained Keras model we selected from TensorFlow was the EfficientNet (EN) B0 Feature Vector model (TensorFlow, 2019). This model was selected over the standard CNN model because these models tend to perform better in image classification tasks (Tan & Le, 2019). An EN model implements a scaling technique that modifies the depth, width, and image resolution dimensions of a network using a compound coefficient (Tan & Le, 2019). The compound coefficient acts a ratio on the three network dimensions which changes under different resource constraints (Tan & Le, 2019). For example, models handling higher resolution images should increase width, adding more channels, in order to capture all the patterns in a larger image (Tan & Le, 2019). Incorporating a compound coefficient balances out the changes between the dimensions and provides a model with optimal network dimensions that achieves state of the art performance.

Implementing this model into our yoga pose classification problem was simple. Before training, yoga pose images were fed into an image generator which rescaled images to a 224x224 pixel format. The images were divided 80% for training and 20% for validation. The resulting

images were 1x224x224x3 tensors and were placed into batches of size 32. The constructed model included a feature extraction layer which incorporated all of the pre-trained model parameters and a dense layer to classify the resulting feature vectors to one of the classes in our yoga pose dataset. During the training process the number of epochs selected was 15. The training and validation accuracy showed impressive results compared to the CNN model. This model was trained under the OpenPose rendered images which showed better results than using yoga pose images directly from the dataset.

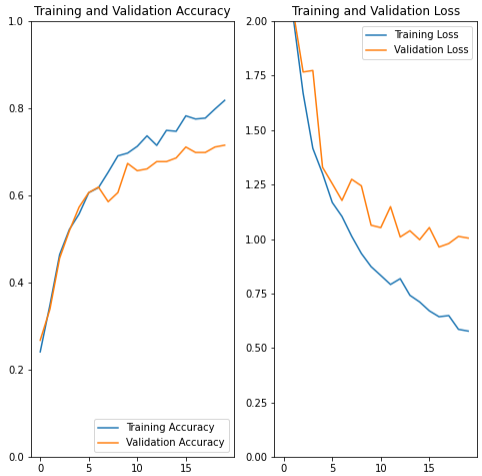
Results

CNN Model

The two graphs below show accuracy and loss for both the training and validation data (Figure 4). The resulting accuracy and loss was recorded for each epoch during the training process. After 20 epochs the model achieved a training accuracy of 81% and a validation accuracy of 71%. The loss for the training data was 0.5775 and validation loss was 1.0045.

Figure 4

CNN model performance graphs



The accuracy from the training and validation data indicate that this model might be overfitting. There was a 10% drop in accuracy when the model was provided with new samples in the testing dataset. The training loss drops significantly while the validation loss remains high. This pattern further highlights the overfitting issues with this model.

EfficientNet Model

The graphs below show the accuracy and loss for the EN model for both the training and validation data (Figure 5). This model was trained over 20 epochs. The training accuracy reached 93.87 % and validation accuracy was 93.19%. Training loss was 0.288 and validation loss was 0.2762.

Figure 5

EfficientNet model performance graphs

The model accuracy in the training and validation datasets show that this model is performing well. Both the training and validation datasets achieve accuracies above 90% and are almost identical. The similarity in accuracy between the training and validation data show that this model is not overfitting as much as the previous CNN model. The loss in the training and

validation data also shows a similar pattern. The loss only differs slightly between training and validation data. The loss is also much smaller than previously seen, meaning that this model is performing better predictions even on unseen data.

Application Results

The examples below show the classification results of the yoga pose application. The application used the EfficientNet model for classification. During training the EfficientNet model showed impressive performance with values above 90% for both training and validation. In the application, the accuracy of the model tends to decrease a bit, as more factors such as camera quality, OpenPose detection, and background affect the new user predictions of the model. The application does a good job in classifying certain poses such as Tree Pose. One of our group members attempted a proper tree pose (Figure 6). Although they forgot to put their leg up, which should suggest the user is doing the pose incorrectly. The Yoga pose application is only 54% confident that the user is doing Tree Pose and tells the user to try again.

Figure 6

Application screenshot of Yoga pose detection for tree pose with 54% confidence

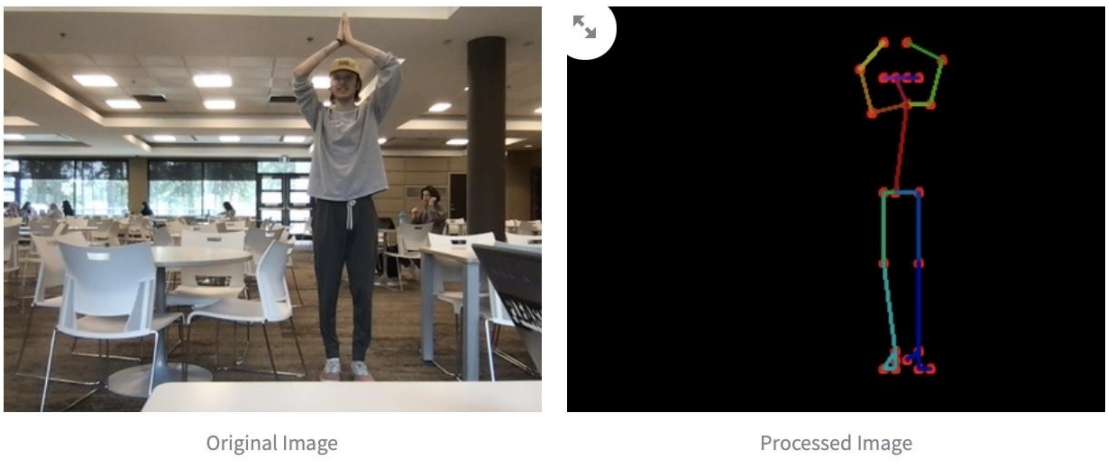
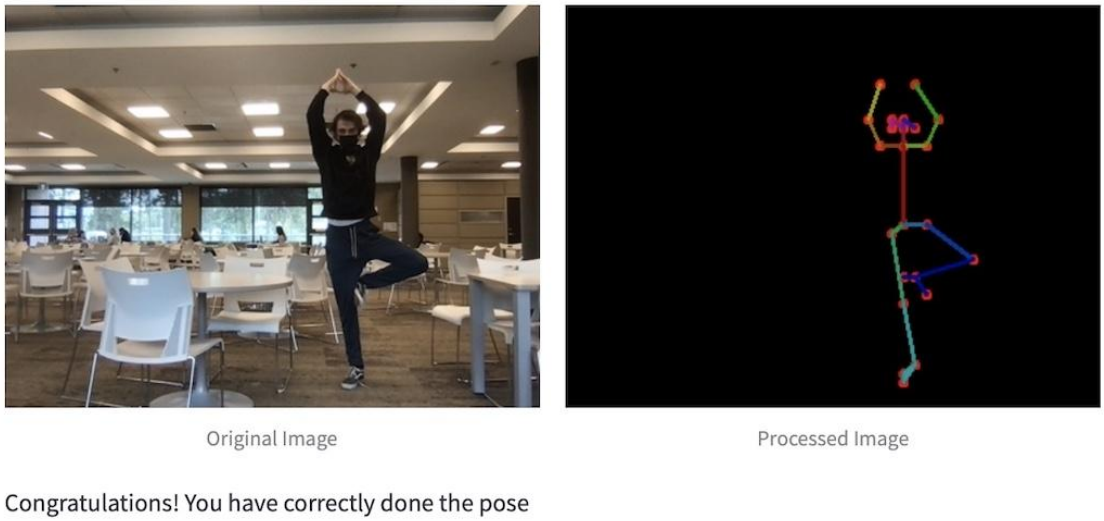


Figure 7

Application screenshot of Yoga pose detection for tree pose with 81% confidence



This result shows how the program is able to distinguish when a pose may be incorrect. Another group member attempts a proper Tree Pose (Figure 7). The yoga pose application has 81% confidence the user is doing Tree Pose and indicates the user has done the pose correctly. The application is able to detect when the user is performing a pose correctly. Goddess is another pose that the application does a good job of classifying as this pose is very simple. Compared to Tree Pose and Goddess, the yoga pose application struggles to provide correct feedback for other poses such as, Cow Face and Fire Log. These poses are more complex and are sometimes mislabeled in OpenPose. Other predictions errors in the model may come from overfitting, small dataset, and user error. Overall the application performs well for simple poses, but fails to detect more complicated positions.

Application Program

The web application, developed using Streamlit, has two main pages: Practice Yoga and About. On the first page, the user is prompted to select one of 14 available Yoga poses to practice. Upon selecting a pose, a sample image is provided so that the user can familiarize themselves with how the pose looks. After selecting a pose, the user has the option to either upload an image or take a picture of themselves doing the pose. After receiving the user's image, the application uses OpenPose to generate a rendered image which is then passed to our modified EfficientNet model to classify it as one of the 14 yoga pose classes. If the user does not specify which yoga pose they would like to practice, but still performs a pose that is able to be detected within the defined accuracy threshold of 0.8, the application will congratulate the user and tell them which pose their image was classified as. If the pose classification does not meet our accuracy threshold, the application will output which pose was detected, and with what accuracy the model detected. However, if the user does specify which pose they are attempting to perform, the system's behavior changes slightly. When the user specifies a pose, and performs it such that the model is able to classify it with an accuracy above the threshold - the user is congratulated. Alternatively, if the user performs a pose that is not their selected pose, but is above the accuracy threshold, the user will be informed that they could potentially be attempting a pose that was not the selected one. In either case, if the pose then does not meet the accuracy threshold, the application will inform the user which pose was detected and the accuracy it was detected with. The application will also suggest that the accuracy is not high enough to be confident in a pose, and that the user should check their form and try again.

Limitations

Despite performing very well overall, there are certain limitations associated with our application. Reliance on third party systems such as OpenPose and StreamLit, although convenient and effective, provide compatibility and functionality challenges which are not easily solved without modifying their source code.

Reliance on OpenPose for the generation of skeletal pose structures means that certain yoga poses such as plank pose, which involve multiple overlapping limbs, are not correctly recognized (Figure 1.1). Therefore, these poses are not able to be implemented into the model as implementing them would result in inaccurate predictions. As well, OpenPose compatibility with MacOS is limited, and can cause issues for some users.

The interfacing aspect of the application is done via the StreamLit framework. Due to the lack of application development experience within the team, StreamLit proved to be the most efficient way to get a working frontend for our application. There are, however, some drawbacks to using a framework such as StreamLit. The primary issues which arise from this are regarding the camera input functionality. StreamLit does not allow for users to set up an image capture timer. This is an issue because in order for a user to take a photo of themselves doing yoga in our application, they must either have somebody around to take the photo for them; or take the photo on an external device and upload it. This does not allow for the user to receive feedback in the seamless way which was envisioned.

Future Improvements

Our first recommendation for improving the application is to have it provide written suggestions and highlight the user's image to help the user improve their pose and prevent injury. To generate suggestions, the application would compare the skeletal angles between the user's

rendered image and the expected angles from the dataset. To ensure that the expected angles are appropriate, this would require a new dataset created by certified instructors to further help prevent injury. The angles and a distance measure (e.g. Euclidean distance) would be used to determine how well the user is performing the pose. Based on individual thresholds for each angle in the skeleton, specific suggestions can be generated. Furthermore, text-to-speech functionality could be implemented so that the user can hear instructions while practicing the pose rather than needing to read the pose, remember the instructions, and attempt the pose away from the device screen.

The second level of improvements would be live video analysis or multiple image uploads with added camera settings. These features are not currently possible with Streamlit, so the user interface would need to be replaced. To implement video analysis, each frame from the video input would be processed and classified. These features would allow the user to perform multiple Yoga poses over time without needing to stop between attempts, but the live video analysis is preferable because it makes the application more interactive and easier to use.

Our final recommended feature is customizable Yoga routines. With a larger dataset, our model could be retrained to classify more poses so that the user can create and edit their own customized Yoga routines. This would further achieve the goal of helping users learn Yoga, and it would increase the likelihood of repeat use of the application.

Conclusion

Pose detection for the purpose of validating yoga pose correctness is a task which both instructors and clients could benefit from. Use of this system would allow for automated generation of Yoga routines, instantaneous feedback on the correctness of poses, and encouragement when poses are done correctly. Increasing the size of the dataset could enable

users to generate more intricate routines, with a degree of difficulty that is better suited to their abilities. Furthermore, addition of live video classification would improve the user experience within the app. If done correctly, live video classification could enable users to receive real time feedback similar to that of which an instructor could provide. However, in order to provide feedback of the quality that an instructor could, it would be necessary for the model to generate suggestions on specific body part positioning. In its current state, the EfficientNet model achieves an ~93% accuracy on both training and validation data - and performs well on real world examples. Overall, we feel that our application is a good first step within this domain, although there is substantial room for improvements along the lines of additional features and enhancements.

References

Andersson, K. & Andreasson, J. (2021). Being a Group Fitness Instructor during the COVID-19 Crisis: Navigating Professional Identity, Social Distancing, and Community. *Social Sciences, 10*(4). doi:10.3390/socsci10040118

Hidalgo, G., Cao, Z., Simon, T., Wei, S.-E., Raaj, Y., Joo, H., & Sheikh, Y. (2022). OpenPose v.1.7.0. https://github.com/CMU-Perceptual-Computing-Lab/OpenPose

Hidalgo, G., Cao, Z., Simon, T., Wei, S.-E., Raaj, Y., Joo, H., & Sheikh, Y. (2022). OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. https://arxiv.org/pdf/1812.08008.pdf

Jain, S., Rustagi, A., Saurav, S., Saini, R., & Singh, S. (2021). Three-dimensional CNN-inspired deep learning architecture for Yoga pose recognition in the real-world environment. *Neural Computing and Applications 33*. 6427–6441. doi:10.1007/s00521-020-05405-5

Lesser, I. A. & Nienhuis, C. P. (2020). The Impact of COVID-19 on Physical Activity Behavior and Well-Being of Canadians. *International Journal of Environmental Research and Public Health, 17*(11). doi: 10.3390/ijerph17113899

Shaw, A. & Kaytaz, E. S. (2021). Yoga bodies, yoga minds: contextualising the health discourses and practices of modern postural yoga. *Anthropology & Medicine, 28*(3). doi:10.1080/13648470.2021.1949943

Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. https://arxiv.org/abs/1905.11946

TensorFlow. (2019). Keras Applications: EfficientNetB0. https://www.tensorflow.org/api_docs/python/tf/keras/applications/efficientnet/EfficientNetB0

Verma, M., Kumawat, S., Nakashima, Y., & Raman, S. (2020). Yoga-82: A New Dataset for Fine-grained Classification of Human Poses. https://arxiv.org/abs/2004.10362