Recognizing Human Actions in Images Using Unsupervised Learning

Naagma Timakondu - Algorithm Engineer

Eli Evans - Software Engineer

Sabrina Hallal - Test Engineer

Applied Cognition and Neuroscience Program, University of Texas at Dallas

Abstract

For technology to assist and respond appropriately to a particular human activity, it may be useful to train models using image snapshots representing actions. This study explored a dataset consisting of images of humans performing different actions and focused on the following actions: calling, running, sitting, sleeping, and texting. Images from the data were re-coded using SVD to compress the images to 30 of the most dominant features. An unsupervised learning model was developed to classify whether the model is familiar or not with the action being depicted in an image. The goal of the unsupervised learning model was to identify the likelihood of the input image of the action falling within the chosen target category. Notably, this study explored the possibility of enhancing the unsupervised learning model by initializing the unsupervised model's weights using weights of a separately trained supervised learning model. The supervised learning model was trained to identify whether images of the action are a part of the target action category or not. In order to identify the best performing model, we varied regularization, amount of data used to train the supervised learning model, and whether or not the unsupervised model was tested on novel actions or not. The model we built struggled to differentiate between images of actions from the target category and those that were images of other actions. In addition, initializing the unsupervised model with the supervised model's weights did not enhance the unsupervised model's learning process. The difficulty in classifying images depicting different human actions by the model built in this study along with other studies highlights the complexity of the problem overall and requires further advanced techniques to solve.

Keywords: Human Actions, image classification, semi-supervised learning, binary cross entropy, clustering

Recognizing Human Actions in Images Using Unsupervised Learning

Image classification continues to be an interesting problem for machine learning algorithms. Identifying images, and more specifically what is being depicted in images, can be useful for a variety of applications. Car systems could use image recognition to identify their surroundings and know, for instance, if someone is cycling next to them versus running next to them. These types of systems could be used in home settings, to provide accurate assistance based on the room they are in and what people in that room are doing. In a hospital setting, this may provide automated assistance based on the situation at hand. Overall, recognizing actions and objects in images can be used to curate appropriate information and provide appropriate responses.

Prior studies have looked at building classification systems for recognizing different human actions using techniques such as Support Vector Machines (SVM) and Hidden Markov models with varying degrees of success. One study used a bag of features classifier and compared that with a structured part based model using the background and foreground contexts to classify human actions (Schuldt et al., 2004). Another study looked at motion patterns to classify actions and used Gaussian convolution kernels, K-means clustering and SVM (Delaitre et al., 2010). Lastly another study utilized a Hidden Markov Model with low-level image features to classify tennis actions (Yamato et al., 1992). However, all of the above studies struggled to build models able to discriminate between actions that consisted of similar features.

In this study, we combined supervised and unsupervised learning algorithms in order to solve the image classification problem of recognizing human actions. The goal was to train an unsupervised model, with the help of the supervised model, to identify whether an input image is depicting someone performing a ‘calling’ action or not. This was done using a binary

cross-entropy log loss function with L2 regularization and batch gradient descent. The expected result is for the model to be able to differentiate between the ‘calling’ and ‘non-calling’ actions. However, as we will discuss later on, the model had trouble differentiating between these actions in the input images. The difficulty of image classification shows the importance of the problem overall and requires further modeling to solve.

Methods

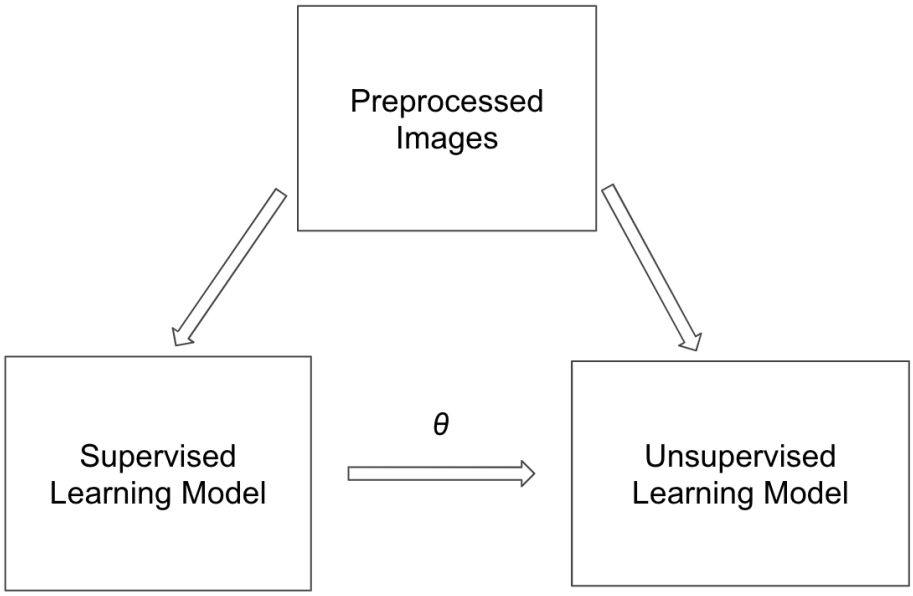
For this project we used the Human Action Recognition dataset available from Kaggle (https://www.kaggle.com/datasets/meetnagadia/human-action-recognition-har-dataset). The dataset consists of about 12,000 images which depict people performing the following actions: calling, clapping, cycling, dancing, drinking, eating, fighting, hugging, laughing, listening to music, running, sitting, sleeping, texting, and using a laptop. For the purposes of this project, we chose five of the actions (calling, texting, sleeping, sitting, and running) and used 842 images from each action category, for a total of 4,210 images. To recode and preprocess these images, we used singular value decomposition and selected the top 30 features of those images to feed as inputs into our model. Each row within the datafile corresponds to an image represented by the top 30 features and the class of the image.

Our model involves use of two algorithms, a supervised learning algorithm and an unsupervised learning algorithm (see Figure 1). The purpose of the model is to take in a set of the pre-processed images and determine whether those images are part of a chosen action category, and produce a binary output to indicate whether the image is a part of that category. For this project, the chosen action category is ‘calling’; the desired output for images from this category is ‘1’, and for images outside of this category (texting, sleeping, sitting, or running) is ‘0’. Calling, sitting, and running images were used for training. An initial split of the training

images are run through the supervised learning algorithm. The training images for the supervised algorithm is further split into 80% training set and 20% test set to ensure the supervised learning machine is performing better than chance. For the unsupervised learning algorithm, only 'calling' images were used, and split into 80% training and 20% test. The unsupervised algorithm is run twice; in one run, the unsupervised learning algorithm takes the trained weights (θ) from the supervised learning algorithm and uses these weights as a 'hint' for the initial guess of the weights it will use, and in the second run the algorithm runs without an initial hint. Both the supervised and unsupervised learning algorithms are tested on the calling, sitting, and running images, as well as novel image categories (texting and sleeping). Both use the same objective function, gradient of the objective function, and gradient descent modules. However, the supervised learning algorithm explicitly uses the action classification of the images to determine whether the image is reflective of the action or not. The unsupervised learning algorithm, on the other hand, takes in the images without their classifications to make its predictions. In summary, the model takes as input a set of unclassified images and returns a binary output, with 1 indicating that the image is depicting the 'calling' action and 0 indicating the image is depicting a 'non-calling' action.

Figure 1

Project Algorithm



The objective function used in the model was the binary cross-entropy log loss function. The mathematical equation for the objective function is

$$\ell_n(\theta) = (1/n) \sum[-y_i \ln(\ddot{y}) - (1-y_i)\ln(1 - \ddot{y})] + \lambda|\theta|^2$$

and can be visualized in Module 1 (see Appendix). The derivative of the objective function was taken to get the gradient of the objective function and L2 regularization was added in this step. The mathematical equation for the gradient is

$$\frac{d\ell_n(\theta)}{d\theta} = (1/n) \sum -(y_i - \ddot{y})[s_i, 1] + 2\lambda\theta$$

as shown in Module 2 (see Appendix). Gradient descent was used as the learning algorithm for this project. The mathematical equation is as follows

$$\theta_{t+1} = \theta_t - \gamma * \frac{dl_n(\theta)}{d\theta}$$

The gradient descent algorithm, as shown in Module 3 (see Appendix), uses stopping criteria of the norm of the gradient being less than epsilon or the number of interactions reaching a

specified maximum. These modules and their code are also found in the appended Python notebook.

The performance of the model was evaluated using several different measures. Firstly, performance of the supervised learning algorithm, unsupervised learning algorithm with a hint, and unsupervised learning algorithm without a hint were compared in terms of the convergence of the gradient norm. The percent of the data that was used to train the supervised learning algorithm was varied, as was the lambda value for the L2 regularization, and results were compared as well. Accuracy, precision, recall, and F1-Scores were calculated for each run, as were within-cluster sum of squares and between cluster sum-of squares. Overall, models were evaluated based on how well they were able to classify 'calling' and 'non-calling' images.

Results

The supervised model was trained and tested on 1%, 5%, and 10% of the calling, sitting, and running actions with three different L2 regularization coefficients of 0 or no regularization, 0.05, and 0.1. The accuracy scores that the supervised model achieved on the train and test set are shown in Table 1, each column representing a different regularization coefficient and each row representing a different percentage of the actions used to train and test the model.

Table 1

Accuracy Scores of Supervised Model

	$\lambda = 0$	$\lambda = 0.05$	$\lambda = 0.1$
Trained on 1%	Train: 80.0% Test: 40.0%	Train: 80.0% Test: 40.0%	Train: 80.0% Test: 40.0%
Trained on 5%	Train: 69.0% Test: 69.2%	Train: 69.0% Test: 69.2%	Train: 69.0% Test: 69.2%
Trained on 10%	Train: 71.6% Test: 72.5%	Train: 71.6% Test: 72.5%	Train: 71.6% Test: 72.5%

The trained thetas from the supervised model were then used as the initial thetas for the unsupervised model. The unsupervised model was trained on every calling action image with the same regularization coefficient used to train the supervised model. The within cluster sum of squares (WCSS) and between cluster sum of squares (BCSS) scores received on calling, running, and sitting actions are reported for each supervised model hint and regularization coefficient in Table 2. Table 3 shows the WCSS and BCSS for each supervised model hint and regularization coefficient tested on calling, sitting, running, texting, and sleeping actions, where the sleeping and texting actions have not been seen before by the unsupervised model or its initialization hint. The ratio between the WCSS and BCSS shown in Tables 2 and 3 are visually shown in Figure 2.

Table 2

Within and Between Cluster Sum of Squares for Unsupervised Model Tested on Known Actions

	$\lambda = 0$	$\lambda = 0.05$	$\lambda = 0.1$
Supervised Hint Trained on 1%	WCSS: 2.09e-05 BCSS: 1.88e-05	WCSS: 1.52e-03 BCSS: 9.96e-04	WCSS: 1.47e-03 BCSS: 9.63e-04
Supervised Hint Trained on 5%	WCSS: 9.09e-05 BCSS: 3.40e-05	WCSS: 1.31e-03 BCSS: 5.15e-04	WCSS: 1.26e-03 BCSS: 4.96e-04
Supervised Hint Trained on 10%	WCSS: 2.61e-04 BCSS: 6.84e-05	WCSS: 1.40e-03 BCSS: 3.66e-04	WCSS: 1.34e-03 BCSS: 3.52e-04

Table 3

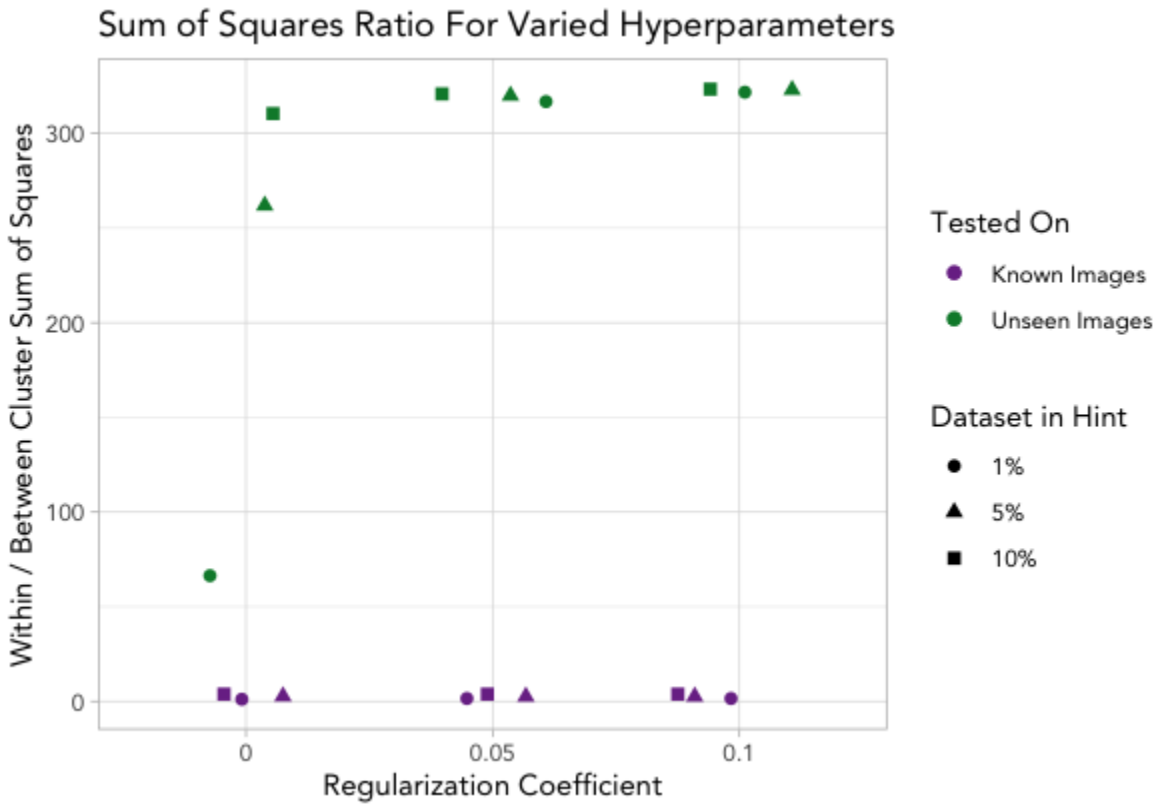
Within and Between Cluster Sum of Squares for Unsupervised Model Tested on Unknown Actions

	$\lambda = 0$	$\lambda = 0.05$	$\lambda = 0.1$
Supervised Hint Trained on 1%	WCSS: 2.41e-05 BCSS: 3.62e-07	WCSS: 1.38e-03 BCSS: 4.36e-06	WCSS: 1.34e-03 BCSS: 4.16e-06
Supervised Hint	WCSS: 9.53e-05	WCSS: 1.39e-03	WCSS: 1.34e-03

Trained on 5%	BCSS: 3.64e-07	BCSS: 4.34e-06	BCSS: 4.14e-06
Supervised Hint Trained on 10%	WCSS: 2.59e-04 BCSS: 8.35e-07	WCSS: 1.39e-03 BCSS: 4.34e-06	WCSS: 1.34e-03 BCSS: 4.14e-06

Figure 2

Clustering Metrics of Unsupervised Learning Machine

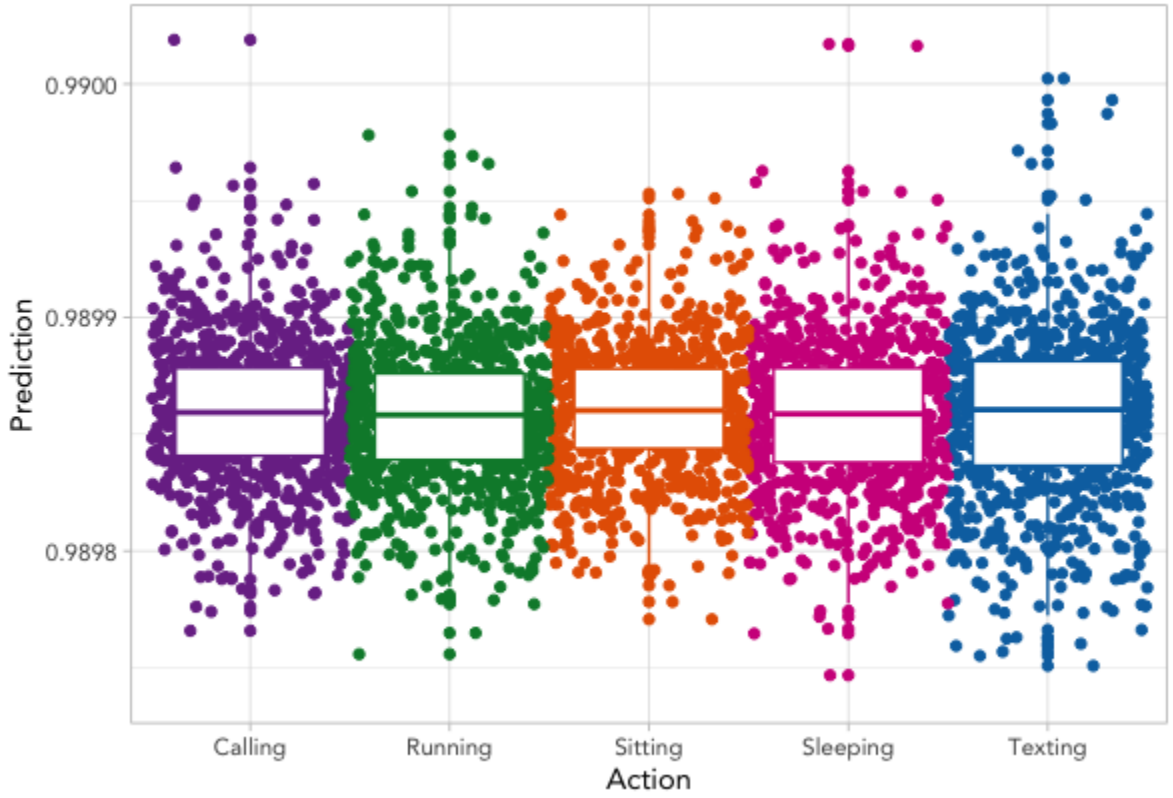


Note. Points are jittered horizontally for visual clarity. Known Images denote images of actions learned by the supervised model, while Unseen Images include both actions learned by the supervised model as well as the texting and sleeping actions, which neither model had been trained on. Dataset in Hint denotes the percentage of actions the supervised model was trained on.

The unsupervised model with the lowest WCSS to BCSS ratio used no regularization and had its thetas initialized from a supervised model using 1% of the calling, sitting, and running actions. This highest-performing model's predictions for each action image is shown in Figure 3 where each point is an image prediction that can be interpreted as the model's stated probability of the image depicting the "calling" action. The greatest prediction average was for the sitting class, followed by calling, texting, sleeping, and running respectively.

Figure 3

Predictions of Unsupervised Learning Machine for Each Action



Note. Points are horizontally jittered to allow visual perceptions of point density. Predictions are shown for the unsupervised model with the lowest within-between cluster sum of squares ratio as shown in Figure 2, with the supervised model being trained on one percent of the dataset and no regularization.

The unsupervised model with thetas initialized with trained thetas from the supervised model consistently performed similarly to an unsupervised model with thetas initialized to zero. The unsupervised model with no regularization and that received the "hint" trained on 1% of the calling, running, and sitting actions received a WCSS of 2.41e-05 and a BCSS of 3.62e-07 on data including unfamiliar actions, while an unsupervised model with the same parameters and thetas initialized to zero received a WCSS of 2.84e-04 and a BCSS of 9.38e-07 when tested on the same data.

Discussion

Overall, the unsupervised model did not perform as well as expected. The model consistently predicted that all of the images it was fed were 'calling' images, and could not differentiate between 'calling' and 'non-calling' images, predicting 'sitting' images as being most likely to belong to 'calling' over the 'calling' action images themselves. Regularization did not seem to aid either model, with the best performing model using no regularization. While the within cluster sum of squares was very low, this was primarily due to all images being predicted very similarly as the between cluster sum of squares was also extremely low, showing almost no difference in predictions between actions. Along with the primary unsupervised model performing poorly, an unsupervised model without a "hint" from the supervised model performed similarly, with a slightly higher BCSS and WCSS, showing greater spread of predictions.

The results showcase the difficulty and limitations of image classification as a whole and, more specifically, in terms of the specific data set chosen. The images themselves tend to be very similar, and the categories are not as clear cut as they may seem, as multiple actions may be portrayed in each image. The model may be having difficulty determining the characteristics of

the actions that differentiate them from one another, especially if the actions have many overlapping characteristics. There is also the question of whether or not the ‘hint’ from the supervised learning algorithm is actually providing useful information to the unsupervised learning algorithm. It may be confusing the algorithm or it may just be altogether not necessary. While other studies used more intricate models, such as variations of Support Vector Machines and Hidden Markov Models, and were able to distinguish between different actions, these models still confused actions that are similar and often misclassified these actions. The shortcoming of this model and the difficulty of image classification as a whole showcases the importance of further developing this and similar models. Afterall, image classification could prove important to developing artificial intelligence, which could serve in a wide variety of environments and circumstances.

Future study is needed to solve the image classification study. More specifically, it is important to understand and improve how models pick out important characteristics among images in order to classify them. Starting with a more clear cut data set, with images that are easily identifiable, will help improve our model itself and would then allow for further development with more complex images. Future studies could also try other regularization terms or methods. Another potential direction for this problem would be to work on different models altogether. These models could improve upon the algorithm we developed, or involve different algorithms, learning rules, loss functions, hidden units, etc. Though our model is a start, there are plenty of ways to further refine it and its utility for machine learning and artificial intelligence.

References

Delaitre, V., Laptev, I., & Sivic, J. (2010). Recognizing human actions in still images: a study of bag-of-features and part-based representations. *BMVC*. doi:10.5244/C.24.97

Schuldt, C., Laptev, I., & Caputo, B. (2004). Recognizing human actions: a local SVM approach, *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004, 3*, 32-36. doi: 10.1109/ICPR.2004.1334462.

Yamato, J., Ohya, J., & Ishii, K. (1992). Recognizing human action in time-sequential images using hidden Markov model. *Proceedings 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 379-385, doi: 10.1109/CVPR.1992.223161.

Appendix

Module 1

Objective Function and Related Code

$$\ell_n(\theta) = (1/n)\,\Sigma[-y_i \ln(\ddot{y}) - (1-y_i)\ln(1-\ddot{y})] + \lambda|\theta|^2$$

```
def phi(si, theta):
    return np.dot(np.append(si, 1), theta)
def ydotdot(si, theta):
    return 1/(1 + np.exp(-phi(si, theta)))
def ci(si, theta, yi):
    return -yi*np.log(ydotdot(si, theta)) - (1-yi)*np.log(1-ydotdot(si, theta))
def ln(s, theta, y, reg_lambda):
    sumci = 0
    for i in range(len(y)):
        sumci += ci(s[i], theta, y[i])
    return sumci/len(y) + reg_lambda*(npnorm(theta, ord=2)^2)
```

Note. For the supervised learning model, y was 0 for non-calling actions and 1 for calling actions, while for the unsupervised model y was always set to 1.

Module 2

Gradient of Objective Function and Related Code

$$\frac{d\ell_n(\theta)}{d\theta} = (1/n)\,\Sigma-(y_i - \ddot{y})[s_i, 1] + 2\lambda\theta$$

```
def gradci(si, theta, yi):
    return -(yi - ydotdot(si, theta))*np.append(si, 1)
def dlndtheta(s, theta, y, reg_lambda):
    sumci = 0
    for i in range(len(y)):
        sumci += gradci(s[i], theta, y[i])
    return sumci/len(y) + reg_lambda*2*theta
```

Note. For the supervised learning model, y was 0 for non-calling actions and 1 for calling actions, while for the unsupervised model y was always set to 1.

Module 3

Gradient Descent Algorithm Function

$$\theta_{t+1} = \theta_t - \gamma * \frac{dl_n(\theta)}{d\theta}$$

$$gradnorm = max(\left|\frac{dl_n(\theta)}{d\theta}\right|)$$

```
def gradientdescent(s, theta, y, reg_lambda=.01, gamma=.01, max_iter=10000):
    gradnorm = np.Inf
    for t in range(max_iter):
        if gradnorm < 10e-4: return theta
        dln = dlndtheta(s, theta, y, reg_lambda)
        theta = np.subtract(theta, gamma*dln)
        gradnorm = np.max(np.abs(dln))
        print("Iterations: "+str(t)+" \t gradnorm = "+str(gradnorm))
    return theta
```