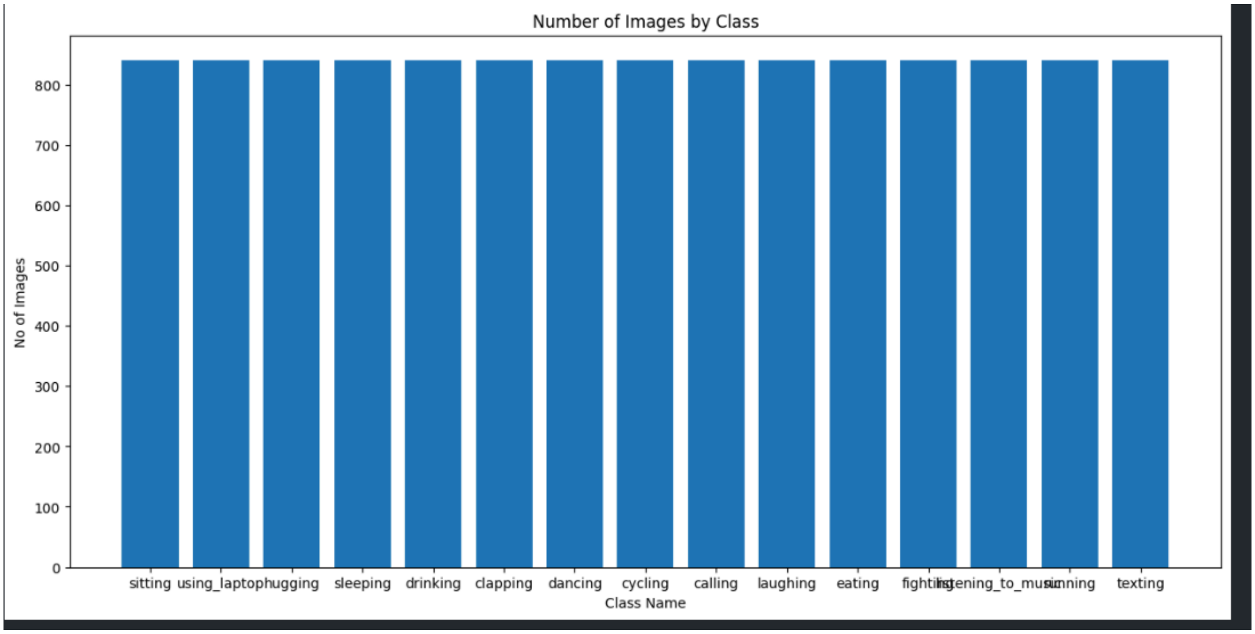
Dataset Overview

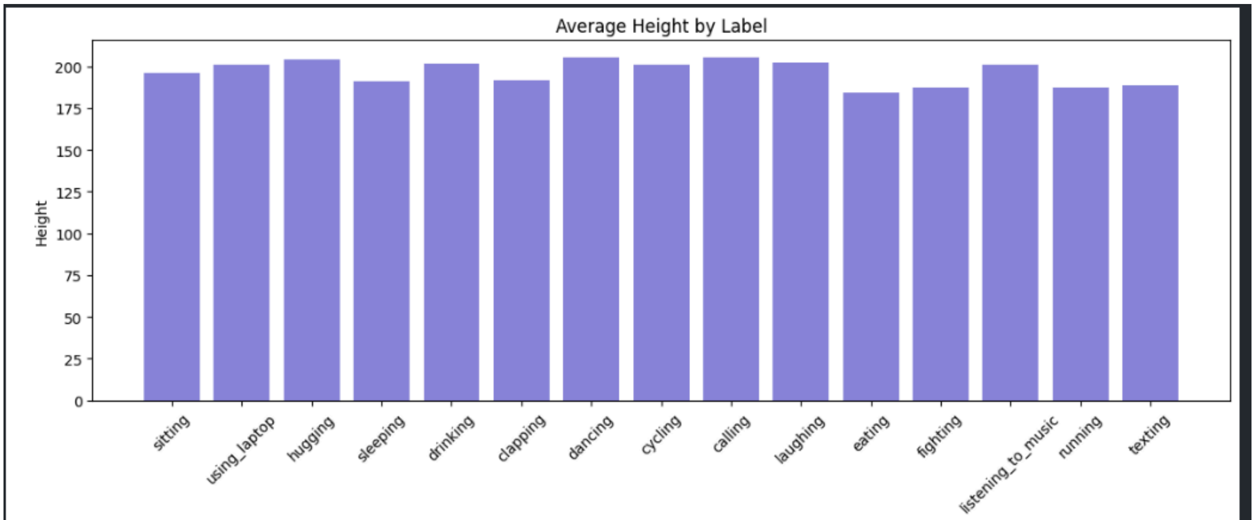
Number of Images per Class:

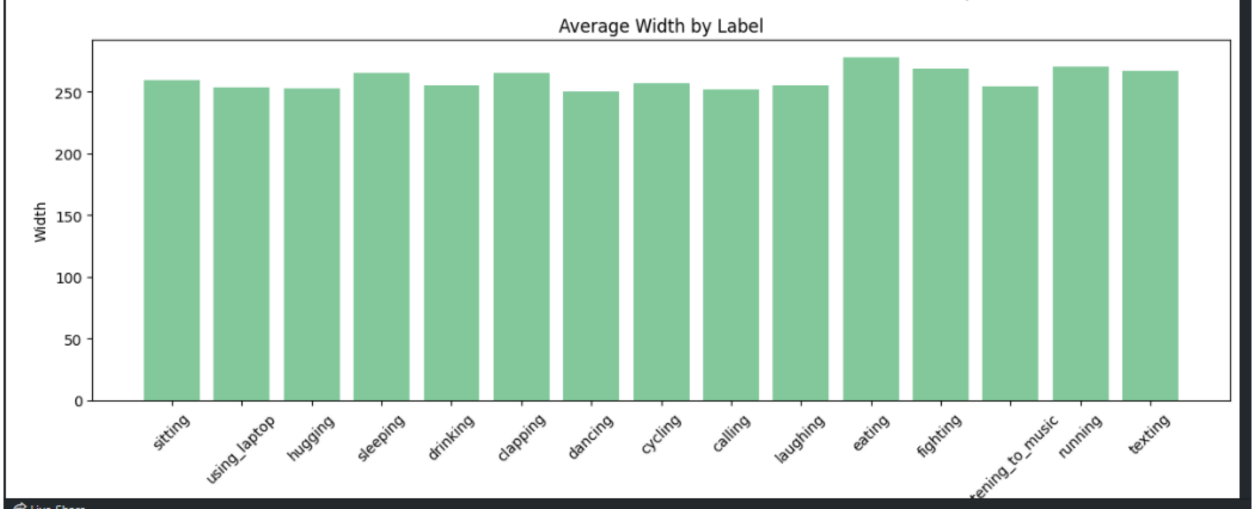
Each class (label) has 840 images, which suggests a balanced dataset. The classes include activities like sitting, using a laptop, hugging, sleeping, drinking, clapping, dancing, cycling, calling, laughing, eating, fighting, listening to music, running, and texting.

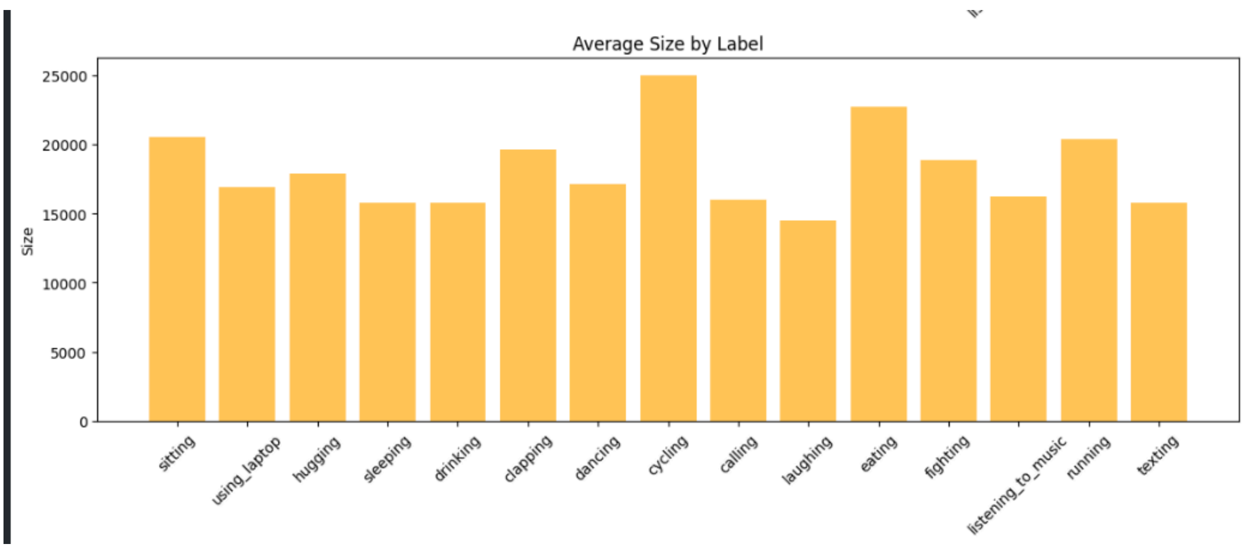


Average Dimensions:

The first two bar charts shows the average height and width of the images for each label. The third chart shows the average size of the images by label.


Average Height by Label
Height
0
25
50
75
100
125
150
175
200
sitting
using_laptop
hugging
sleeping
drinking
clapping
dancing
cycling
calling
laughing
eating
fighting
listening_to_music
running
texting


Average Width by Label
Width
0
50
100
150
200
250
sitting
using_laptop
hugging
sleeping
drinking
clapping
dancing
cycling
calling
laughing
eating
fighting
running
texting


Average Size by Label
Size
0
5000
10000
15000
20000
25000
sitting
using_laptop
hugging
sleeping
drinking
clapping
dancing
cycling
calling
laughing
eating
fighting
listening_to_music
running
texting

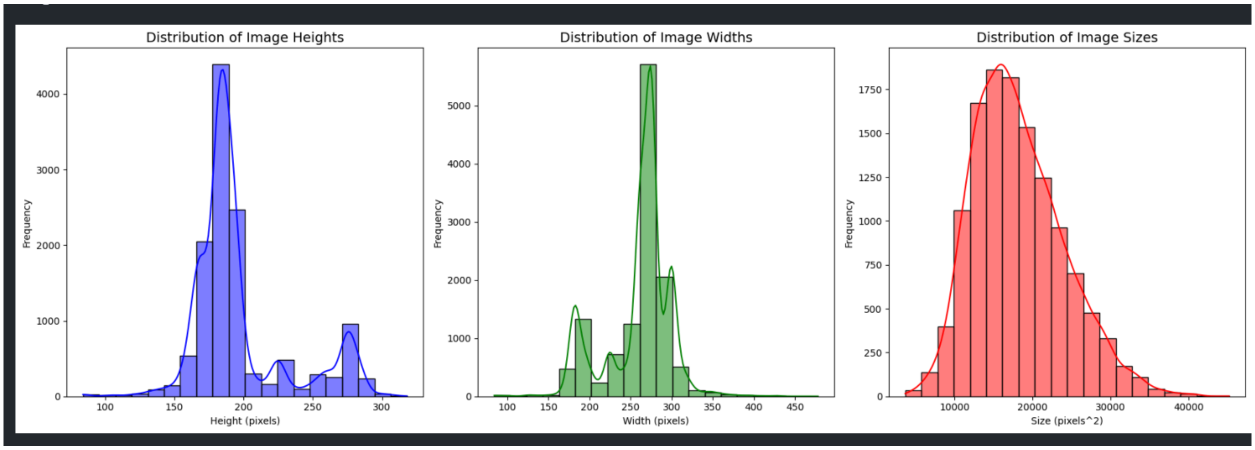
Statistics:

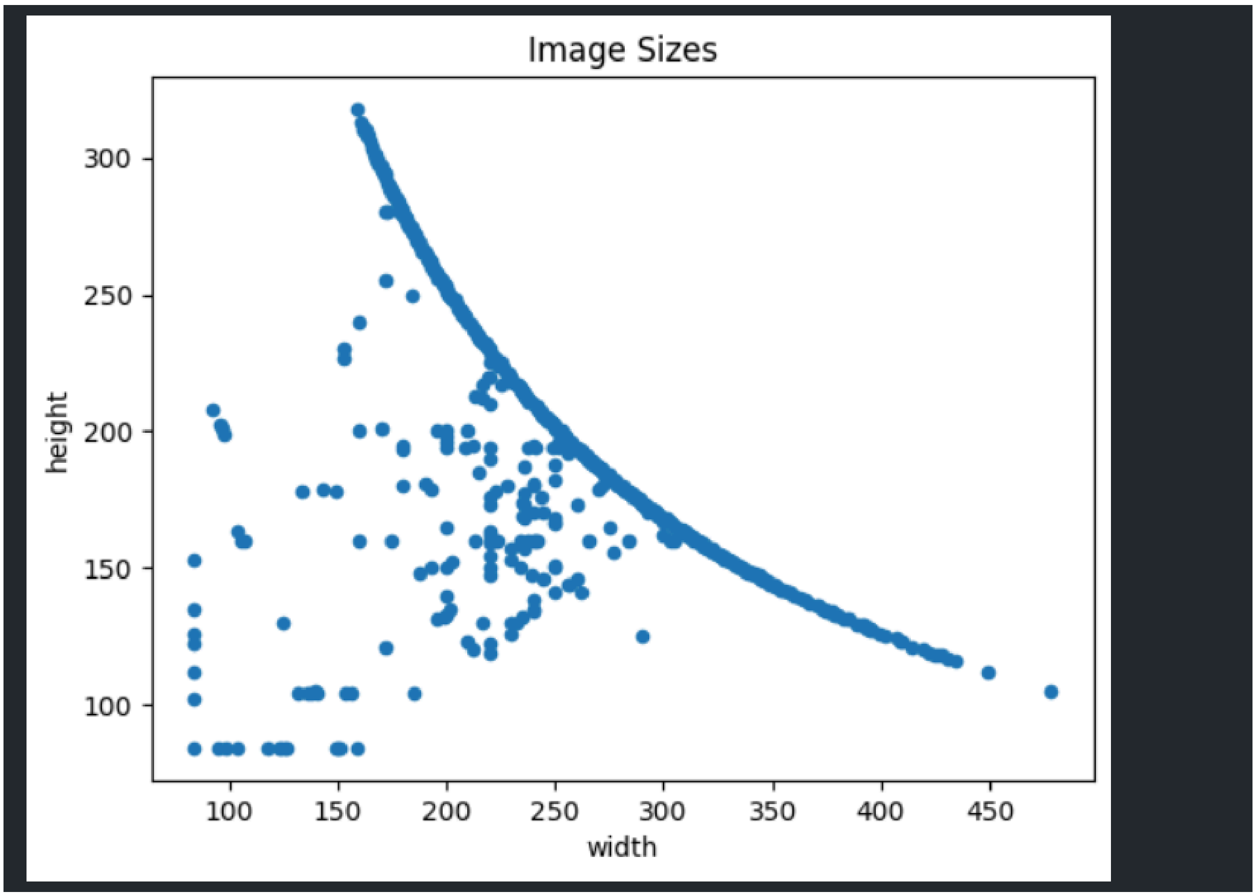
The height and width shows a consistent average across different labels denoting that the images are likely standardized in terms of dimensions.
The average size of the images shows some variability, with cycling having notably higher and laughing being the lowest compared to other labels.

Observations
The dataset is well-balanced in terms of the number of images across all classes, which is crucial for training machine learning models. This helps prevent bias toward any particular class.

The consistent average dimensions across labels suggest that the dataset is likely designed to capture specific activities in similar settings, which can improve the model's ability to generalize from training to validation/test data.

The variation in average image size indicates that some classes may contain images with higher detail or resolution. For instance, the higher average sizes for "cycling" and "eating" could be due to more complex scenes or a greater number of objects being depicted.





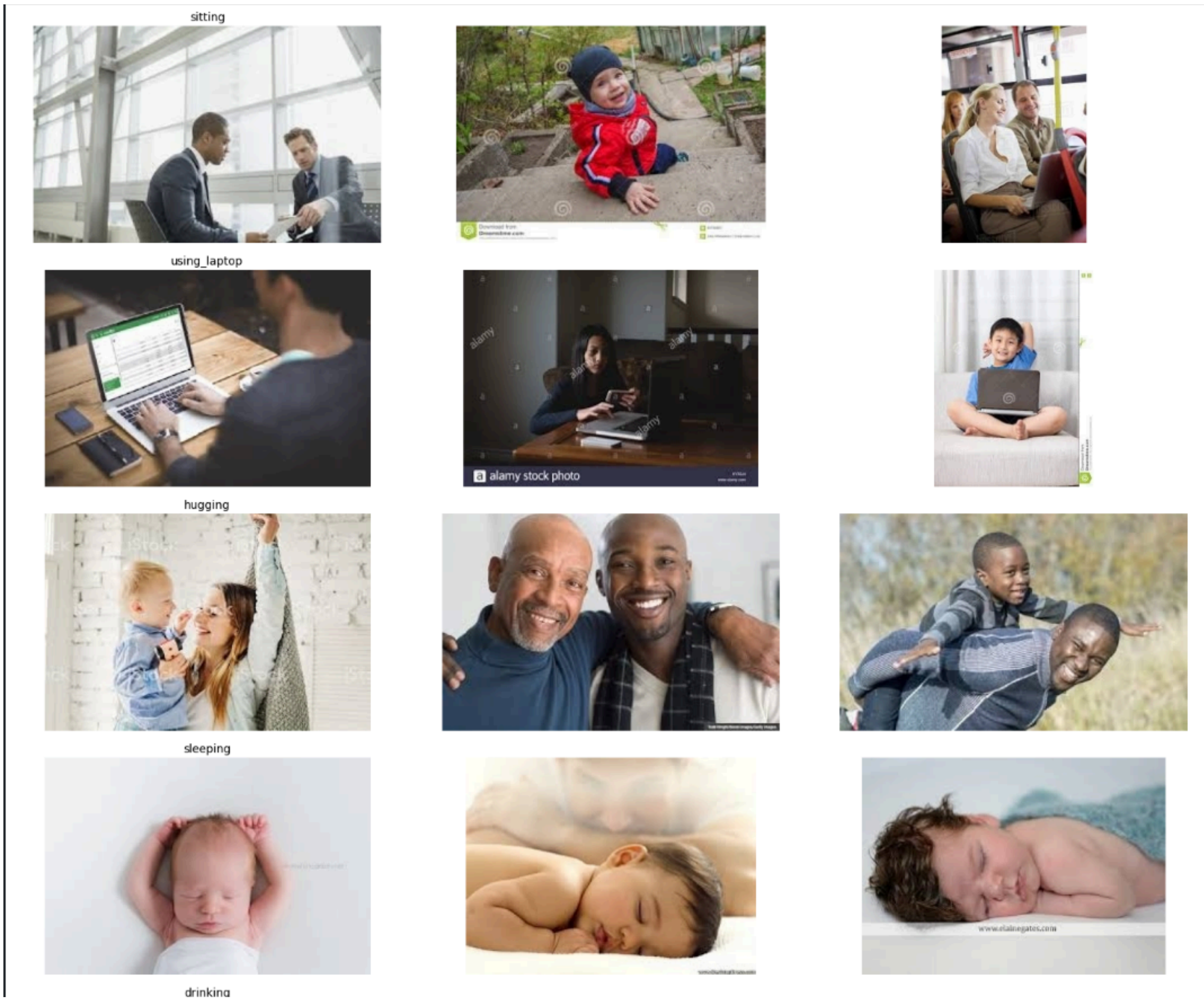
This plot reveals the correlation between image height and width. We can observe a clear negative curve trend, where larger widths correspond to smaller heights or vice versa.
Most of the images had width between 250 and 300 and height between 170 to 200 pixels.
The sizes are roughly normally distributed, peaking around 15,000 . This shows the average image size, with most images falling between 10,000 to 25,000 .

The bar chart titled "Number of Images by Class" shows that the class distribution is perfectly balanced, with each class (e.g., sitting, using a laptop, hugging, etc.) having an equal number of images, specifically 840. Since every class has the same number of images, there is no class imbalance present in the dataset This is ideal for training machine learning models because it ensures that the model is not biased toward any particular class due to an overrepresentation or underrepresentation of data.

Strategies (if there were imbalances)

Although there is no imbalance, here are some strategies that could be used if imbalances were present:

We could have used strategies like data augmentation (like rotations ,flipping), Oversampling like SMOTE (increase the representation of minority classes by generating synthetic samples that are similar to existing ones.).Undersampling could be used to reduce the number of images in those classes to match the minority classes.


sitting
using_laptop
alamy stock photo
hugging
sleeping
drinking

drinking



clapping

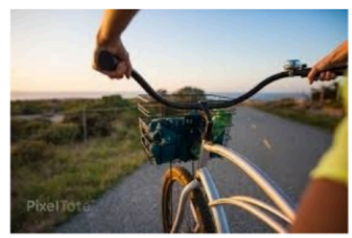


dancing

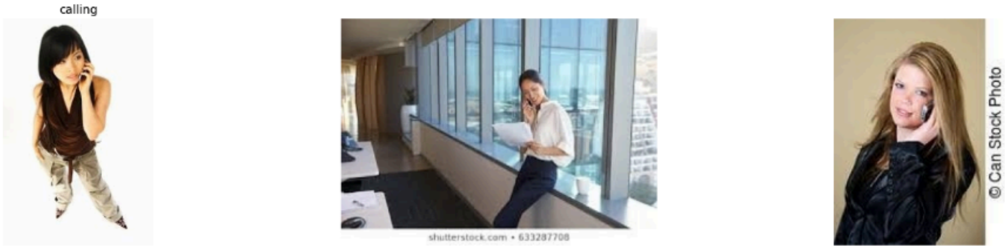


cycling





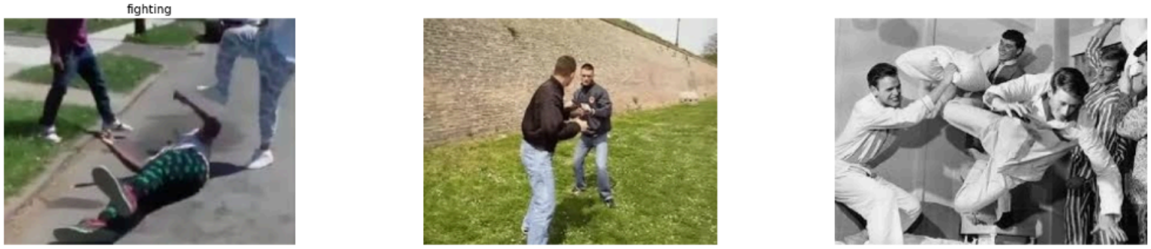
calling

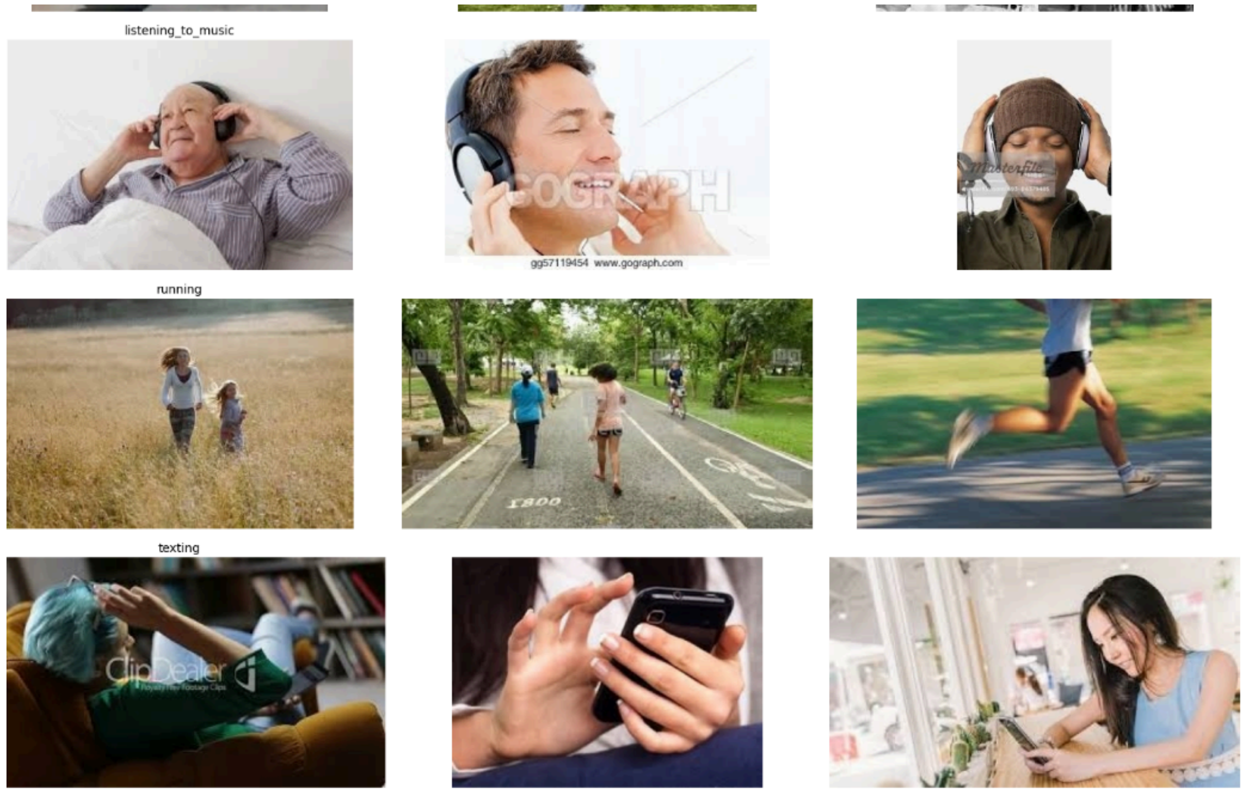


laughing

eating

fighting





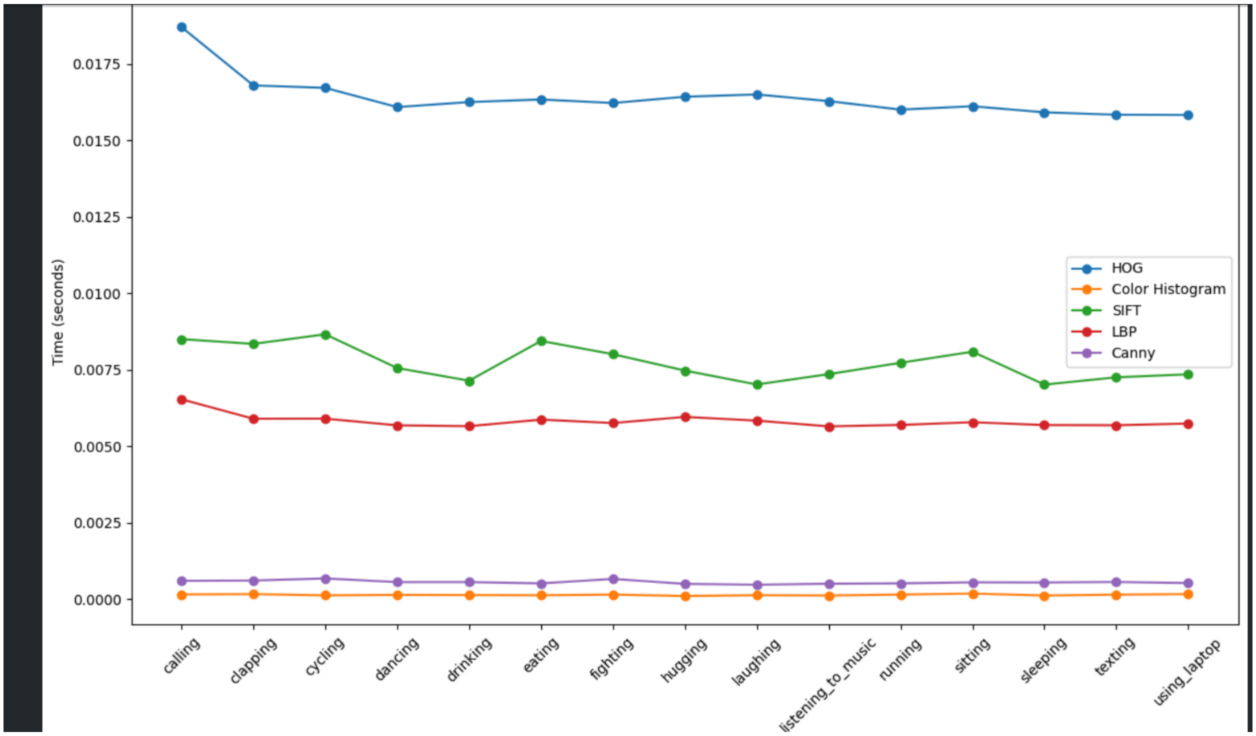
B)-Feature Extraction Methods
L1 l2 l3
HOG (Histogram of Oriented Gradients) Features: Captures the gradient structure of an image by computing histograms of gradient directions over small regions, Good at capturing shapes; effective for classes where shape is a key distinguishing feature.

Color Histogram: This represents the distribution of colors in an image using a 3D histogram of RGB color intensities.

SIFT (Scale-Invariant Feature Transform) Features:It extracts features based on detecting keypoints and extracting local feature descriptors. SIFT features are invariant to scale, orientation, and affine transformations

LBP (Local Binary Pattern) Features: Analyzes local textures by thresholding surrounding pixels and encoding the result as binary patterns, Captures texture, effective for classes with significant texture differences.

Canny Edge Features: Detects edges in an image by identifying areas of significant intensity change using a multi-stage algorithm. Highlights object outlines, useful for classes with well-defined edges.
Accuracy comparison - hog + histo for perceptron and hog + histo +lbp for random forest



The above graph compares the average time of extracting features for each label with 200 samples.The average time to extract is maximum for HOG , and the lowest for color histogram.

C)-

Models used -
Random forest with each pair of combination and hyperparamter tuning to get the result -
Best Combination of Features: ('HOG', 'HISTO', 'LBP')
Best Accuracy: 0.3563
Best Hyperparameters: {'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 300}
{'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 300} : 30 smote pca
{'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 300} : 0.33 smote

```
feature_sets = {
    'HOG': hogFeatures_np,
    'HISTO': histoFeatures_np,
    'SIFT': siftFeatures_np,
    'LBP': lbpFeatures_np,
    'GABOR': gaborFeatures_np
```

```
}
```

```
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

With perceptron different combination -
[(('HOG',), 0.20277777777777778),
 (('HISTO',), 0.14642857142857144),
 (('SIFT',), 0.08015873015873017),
 (('LBP',), 0.11785714285714285),
 (('GABOR',), 0.06507936507936508),
 (('HOG', 'HISTO'), 0.2400793650793651),
 (('HOG', 'SIFT'), 0.07896825396825397),
 (('HOG', 'LBP'), 0.175),
 (('HOG', 'GABOR'), 0.18571428571428572),
 (('HISTO', 'SIFT'), 0.07817460317460317),
 (('HISTO', 'LBP'), 0.13174603174603175),
 (('HISTO', 'GABOR'), 0.1638888888888889),
 (('SIFT', 'LBP'), 0.07857142857142857),
 (('SIFT', 'GABOR'), 0.07817460317460317),
 (('LBP', 'GABOR'), 0.10674603174603174),
 (('HOG', 'HISTO', 'SIFT'), 0.0873015873015873),
 (('HOG', 'HISTO', 'LBP'), 0.1496031746031746),
 (('HOG', 'HISTO', 'GABOR'), 0.22857142857142856),
 (('HOG', 'SIFT', 'LBP'), 0.08492063492063492),
 (('HOG', 'SIFT', 'GABOR'), 0.07896825396825397),
 (('HOG', 'LBP', 'GABOR'), 0.225),
 (('HISTO', 'SIFT', 'LBP'), 0.0761904761904762),
 (('HISTO', 'SIFT', 'GABOR'), 0.07817460317460317),
 (('HISTO', 'LBP', 'GABOR'), 0.15158730158730158),

For any of the hyperparamter perceptron_model = Perceptron(max_iter=1000, tol=1e-3, random_state=42) were giving the best result . Got the result after doing hyperparameter tuning.
for perceptron
best-(max_iter=1000, tol=1e-3) : 23

```
param_grid = {
    'max_iter': [1000, 2000, 3000],
    'tol': [1e-3, 1e-4]
}
```

Best Combination of Features: ('HOG', 'HISTO')
Best Accuracy: 0.2401
Best Hyperparameters: {'max_iter': 1000, 'tol': 0.0001}
the accuracy of perceptron increased while increasing the n components from 50 to 300
Applying PCA with 50 components
Perceptron Model Accuracy with 50 PCA components: 0.1357142857142857
Applying PCA with 100 components
Perceptron Model Accuracy with 100 PCA components: 0.15674603174603174
Applying PCA with 150 components
Perceptron Model Accuracy with 150 PCA components: 0.18968253968253967
Applying PCA with 200 components
Perceptron Model Accuracy with 200 PCA components: 0.19563492063492063
Applying PCA with 250 components
Perceptron Model Accuracy with 250 PCA components: 0.19642857142857142
Applying PCA with 300 components
Perceptron Model Accuracy with 300 PCA components: 0.2123015873015873

On hog , histo features -
Decision Tree Model Accuracy: 0.16150793650793652
Naive Bayes Model Accuracy: 0.12420634920634921
Perceptron Model Accuracy: 0.23095238095238096

Ensemble models-
AdaBoost Model Accuracy: 0.25277777777777777 (hogFeatures_np, histoFeatures_np,lbpFeatures_np)

Best accuracy -
Using voting classifier with random forest and xgbClassifier **0.4126984126984127**

This result shows the strength of ensemble methods in improving predictive performance by combining multiple models.
The Decision Tree and Naive Bayes models performed poorly in comparison to other models.
The Perceptron model showed moderate accuracy, with the highest accuracy achieved with the HOG and HISTO feature combination.

Accuracy improved when increasing PCA components, showing the benefits of feature dimensionality reduction for this model.
The Random Forest model performed best with a feature combination of HOG, HISTO, and LBP.