Machine Learning Techniques for Distracted Drivers Detection

GROUP MEMBERS

TAJ MOHAMMAD (20535030)

KM KHUSHBU (20535014)

VATSAL TIWARI (20535032)

INTRODUCTION

- Distracted driving is a main factor that cause severe car accidents. It has been suggested as a possible contributor to the increase in fatal crashes from 2014 to 2018 and is a source of growing public concern.
- This project focuses on **driver distraction activities detection** via images, which is useful for vehicle accident precaution. We aim to build a high accuracy classifiers to distinguish whether drivers is driving safely or experiencing a type of distraction activity.

DATASET

- The training dataset contains **22424 images** categorized in **10 classes** from *StateFarm*.

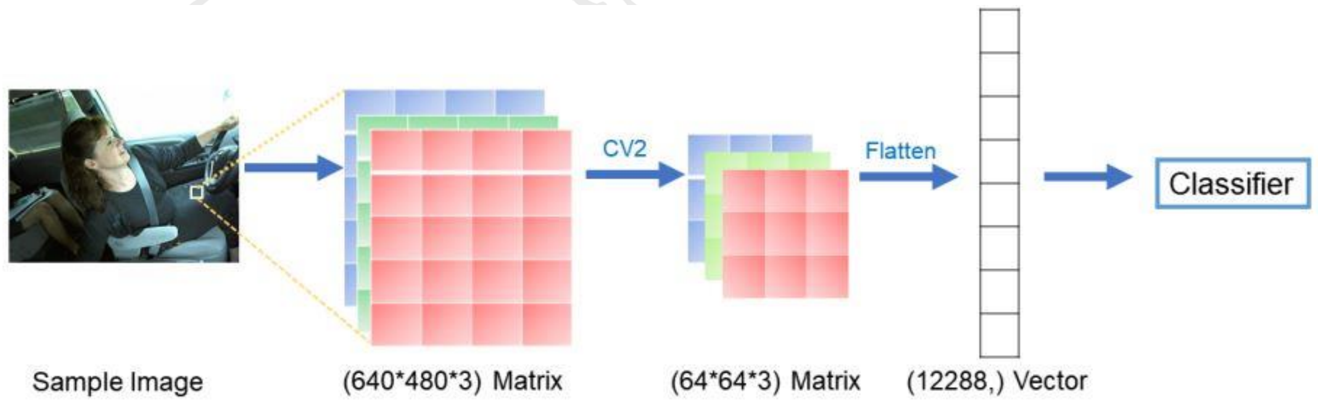
- We randomly split the dataset into two folds: **80% for training**, **20% for validation**.

- One category represents safety driving, and other 9 categories represents 9 different distraction activities we consider here.

Class Symbol	Class Name	Number of Training Images	Sample Class Image
C0	Normal Driving	2490	
C1	Texting (right)	2268	
C2	Talking on the phone (right)	2318	
C3	Texting (left)	2347	
C4	Talking on the phone (left)	2327	
C5	Operating the radio	2313	
C6	Drinking	2326	
C7	Reaching behind	2003	
C8	Hair and Makeup	1912	
C9	Talking to passenger	2130	

PREPROCESSING

- Images in the dataset have very high resolutions**(640×480×3).**
- In order to improve the computational efficiency, we preprocessed the images by resizing them to **(64×64×3)**.
- flattened the high dimensional image matrix to image vectors as the input to train the classifiers.




```
def resize(path, img_height, img_width):
    img = cv2.imread(path)
    resized = cv2.resize(img, (img_height, img_width))
    return resized
```

```
for fl in files:
    flbase = os.path.basename(fl)
    img = resize(fl, 64, 64)
    X.append(img)
```

```
X = np.reshape(X, (X.shape[0], -1))
```

MODELS

Existing Models

- Linear Support Vector Machine(SVM) classifier
- Naïve Bayes Classifier
- Two-layer Neural Network
- Softmax Regression

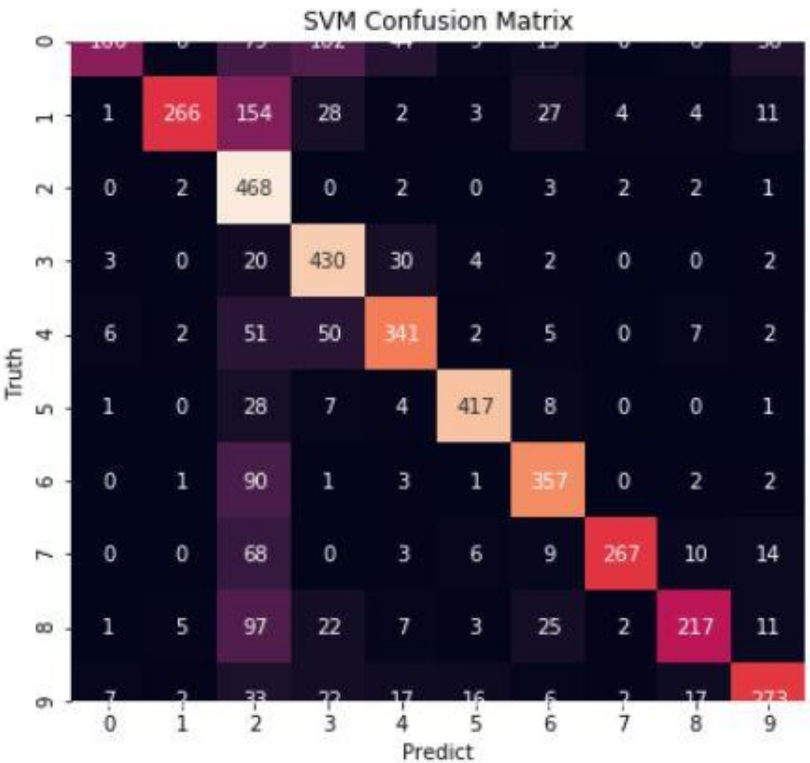
Novel Approach

- K-Nearest Neighbors(KNN) Classifier
- Random Forest Classifier
- Convoluted Neural Networks(CNN)

Existing Models

- **Linear SVM Classifier**

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i,W)_j - f(x_i,W)_{y_i} + 1) + \lambda\|W\|_2^2$$



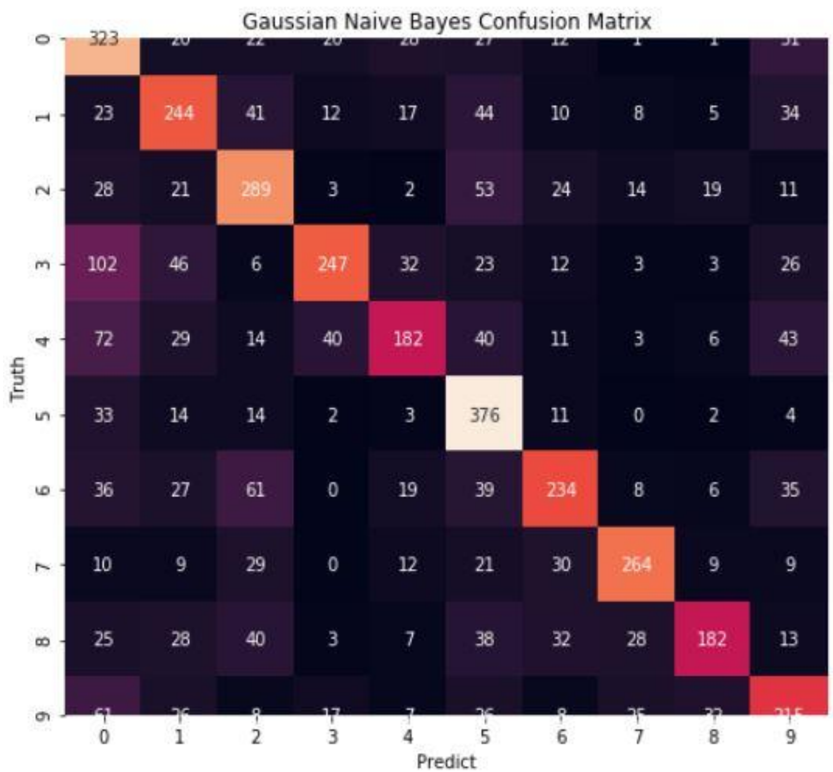
SVM Performance Matrics

	precision	recall	f1-score	support
0	0.90	0.36	0.51	463
1	0.94	0.53	0.68	500
2	0.43	0.97	0.60	480
3	0.65	0.88	0.75	491
4	0.75	0.73	0.74	466
5	0.90	0.89	0.90	466
6	0.78	0.78	0.78	457
7	0.96	0.71	0.82	377
8	0.82	0.56	0.66	390
9	0.77	0.69	0.73	395
accuracy			0.71	4485
macro avg	0.79	0.71	0.72	4485
weighted avg	0.79	0.71	0.71	4485

Existing Models

- **Naïve Bayes Classifier**

$$P(x_i \mid c_k) = \frac{1}{\sqrt{2\pi\sigma^2_{c_k}}} \exp\left(-\frac{(x_i - \mu_{c_k})^2}{2\sigma^2_{c_k}}\right) \qquad \hat{c_k} = \underset{c_k}{\operatorname{argmax}}\, P(c_k) \prod_{i=0}^{n} P(x_i \mid c_k)$$



Gaussian Naive Bayes Performance Matrix

	precision	recall	f1-score	support
0	0.45	0.64	0.53	505
1	0.53	0.56	0.54	438
2	0.55	0.62	0.59	464
3	0.72	0.49	0.59	500
4	0.59	0.41	0.49	440
5	0.55	0.82	0.66	459
6	0.61	0.50	0.55	465
7	0.75	0.67	0.71	393
8	0.69	0.46	0.55	396
9	0.49	0.51	0.50	425
accuracy			0.57	4485
macro avg	0.59	0.57	0.57	4485
weighted avg	0.59	0.57	0.57	4485

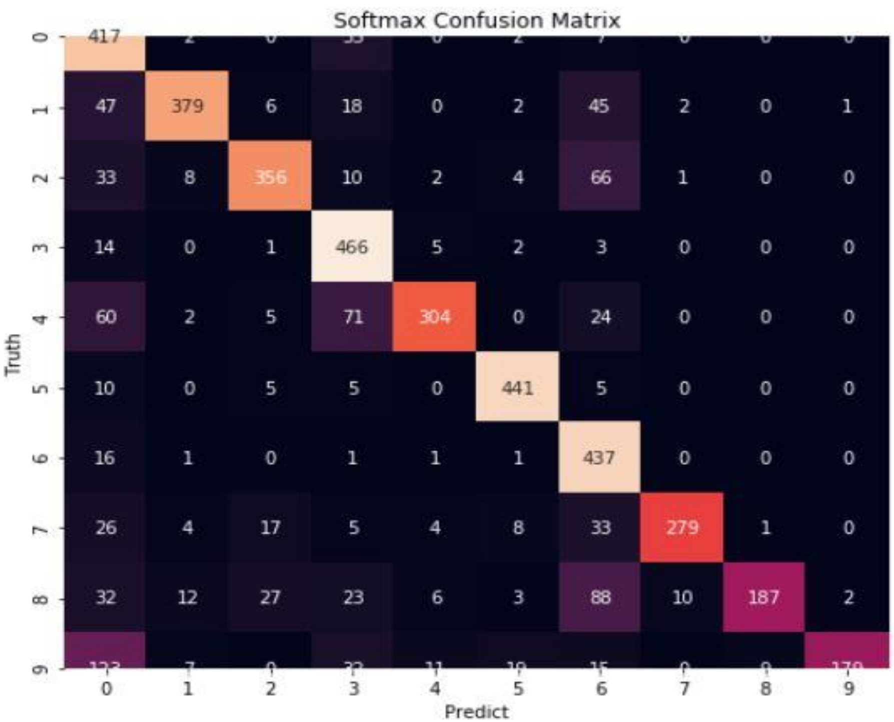
Existing Models

- **Softmax Regression**

Softmax function

$$P(y=j \mid \Theta^{(i)}) = \frac{e^{\Theta^{(i)}}}{\sum_{j=0}^{k} e^{\Theta_k^{(i)}}}$$

where $\Theta = w_0x_0 + w_1x_1 + \ldots + w_kx_k = \sum_{i=0}^{k} w_ix_i = w^Tx$

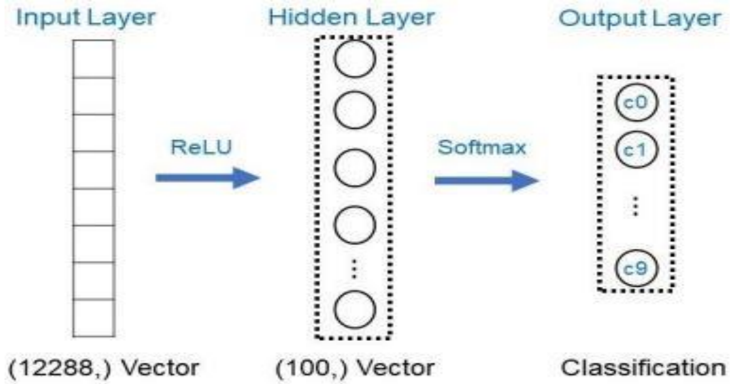


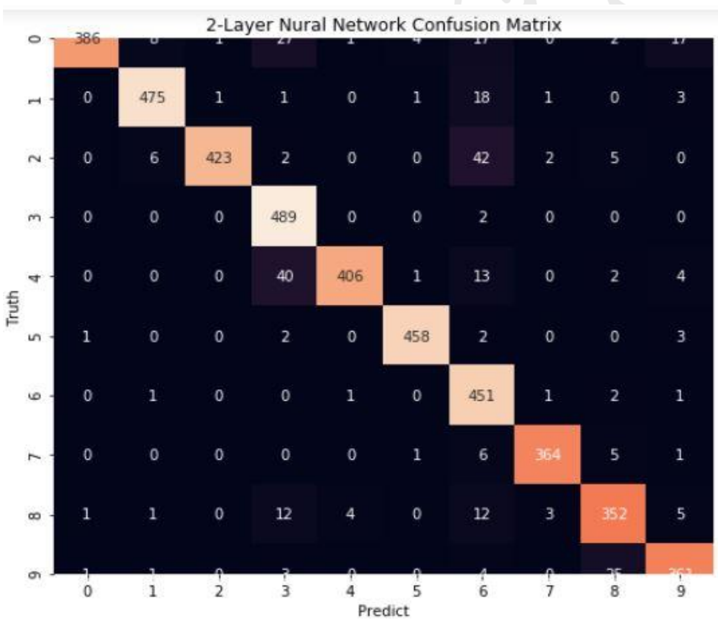
softmax Performance Matrix

	precision	recall	f1-score	support
0	0.54	0.90	0.67	463
1	0.91	0.76	0.83	500
2	0.85	0.74	0.79	480
3	0.70	0.95	0.81	491
4	0.91	0.65	0.76	466
5	0.91	0.95	0.93	466
6	0.60	0.96	0.74	457
7	0.96	0.74	0.83	377
8	0.95	0.48	0.64	390
9	0.98	0.45	0.62	395
accuracy			0.77	4485
macro avg	0.83	0.76	0.76	4485
weighted avg	0.83	0.77	0.77	4485

Existing Models

- **Artificial Neural Network(ANN)**





2-Layer Nural Network Performance Matrics

	precision	recall	f1-score	support
0	0.99	0.83	0.91	463
1	0.97	0.95	0.96	500
2	1.00	0.88	0.93	480
3	0.85	1.00	0.92	491
4	0.99	0.87	0.92	466
5	0.98	0.98	0.98	466
6	0.80	0.99	0.88	457
7	0.98	0.97	0.97	377
8	0.90	0.90	0.90	390
9	0.91	0.91	0.91	395
accuracy			0.93	4485
macro avg	0.94	0.93	0.93	4485
weighted avg	0.94	0.93	0.93	4485

Improvement over the existing models:

1. Principal component analysis (PCA):

- The images are resized to ($64 \times 64 \times 3 = 12288$) ,here Individual pixel used as a feature and dimensionality of the data is too large so computational complexity is very high and ML models takes long time to process this 12288 dimensionality data

- So to solve this problem we use PCA for dimensionality Reduction.
- Number of features Reduced from 12288 to 507 on retaining 95% variance.
- Thus ,we reduces the computational complexity as well as overfitting.

Improvement over the existing models:

1. Principal component analysis (PCA):

```
from sklearn.decomposition import PCA
pca = PCA(0.95)
```

```
pca.fit(X_train)
```

```
PCA(copy=True, iterated_power='auto', n_components=0.95, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

```
X_train_pca = pca.transform(X_train)
X_validate_pca = pca.transform(X_validate)
```

```
print("Number of Features Before PCA")
m,n= X_train.shape
print(n)
print()
print("Number of Features After PCA with 95% variance")
m,n= X_train_pca.shape
print(n)
```

```
Number of Features Before PCA
12288

Number of Features After PCA with 95% variance
507
```

Improvement over the existing models:

1. Principal component analysis (PCA):

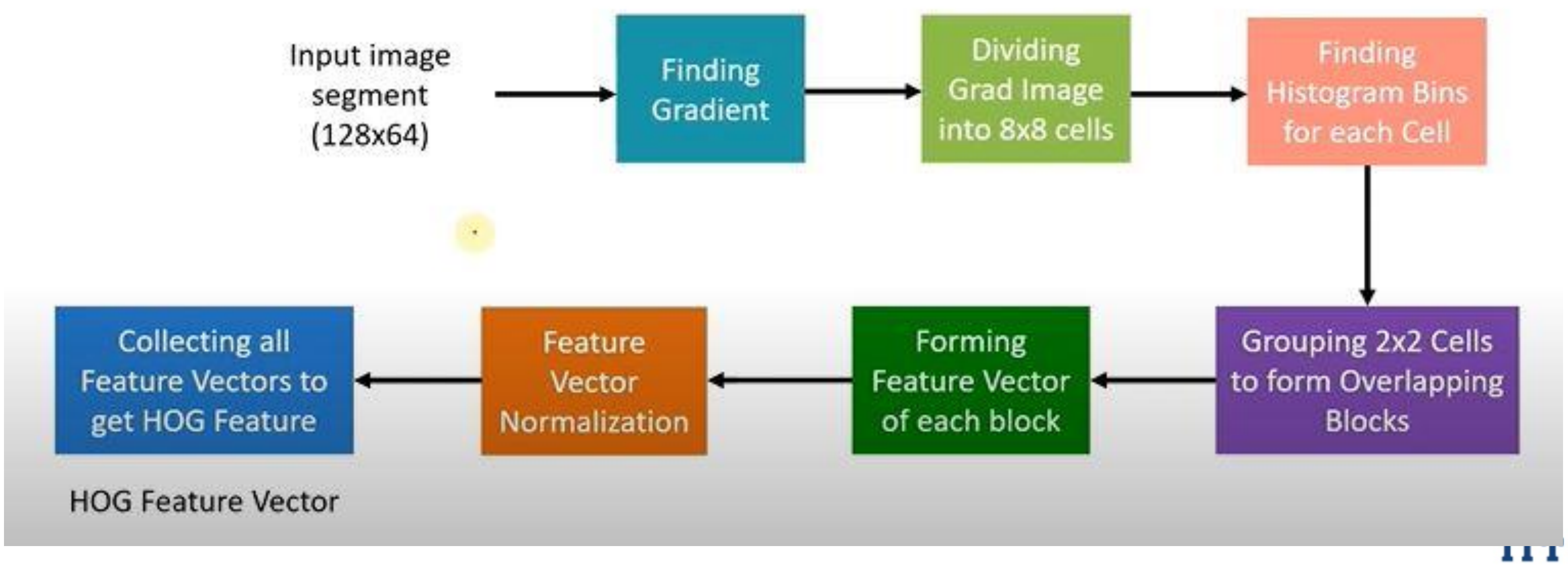
Model	Validation Set Accuracy	Validation Set Accuracy with PCA
Linear SVM	71.39	70.33
Gaussian Naïve Bayes	56.98	55.11
Two-layer Neural Network	92.86	92.92
Softmax Regression	76.81	77.20

Improvement over the existing models:

2. Histogram of oriented gradients:

The Histogram of Oriented Gradients(HOG) is a feature descriptor that is used in computer vision and image processing for the purpose of object detection.

Process:



Improvement over the existing models:

HOG Feature Descriptor:

1. Count occurrences of gradient orientation in localized portions
2. Stacked HOG gradient features to generate a feature matrix

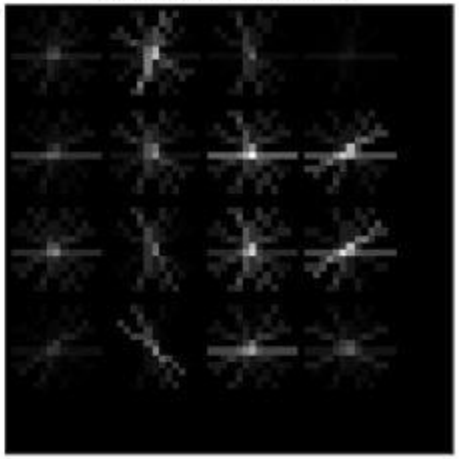
```
from skimage.feature import hog
ppc = 16
hog_images = []
hog_features = []
for image in data_gray:
    fd,hog_image = hog(image, orientations=9, pixels_per_cell=(ppc,ppc),
                       cells_per_block=(2, 2),block_norm='L2-Hys',visualize=True)
    hog_images.append(hog_image)
    hog_features.append(fd)
```

```
print("Original Feature Matrix","     ","HOG Feature Matrix")
print("    ",X.shape,"              ",hog_features.shape)
```

```
Original Feature Matrix      HOG Feature Matrix
    (22424, 12288)               (22424, 324)
```

Original_Image

HOG_Feature_Image



Improvement over the existing models:

- **Existing Models**

Model	Validation Set Accuracy	Validation Set Accuracy with HOG
Linear SVM	71.39	86.33
Multinomial Naïve Bayes	39.38	45.26
Gaussian Naïve Bayes	56.98	54.02
Two-layer Neural Network	92.86	93.12
Softmax Regression	76.81	78.20

Improvement over the existing models:

3. Extended Existing Models from Images to Video:

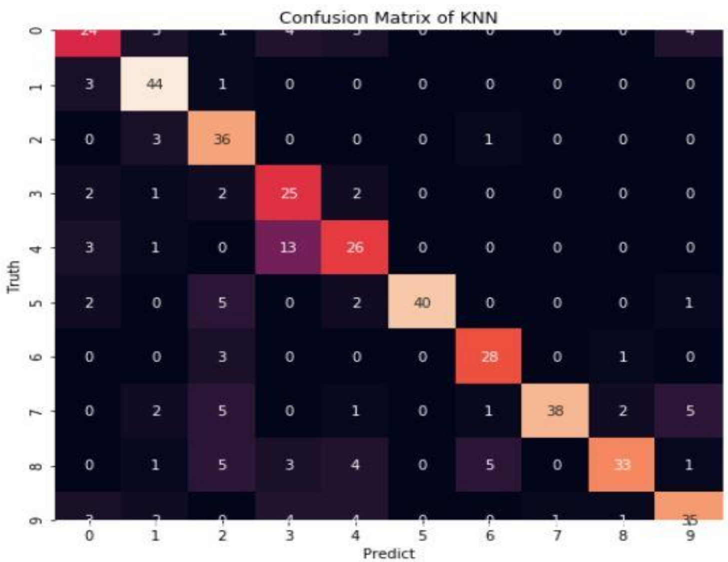
- Using VideoCapture( ) Method of cv2 we captures the video frames.
- Resize each frame into (64*64*3) using resize( ) method of cv2.
- Now Flatten the frame into a vector of (12288,) using reshape( ) method and give it to a Trained Existing model and model will classified it.

Novel Approach

- **K Nearest Neighbours (KNN) Classifier**

$$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + \cdots + (q_n-p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$



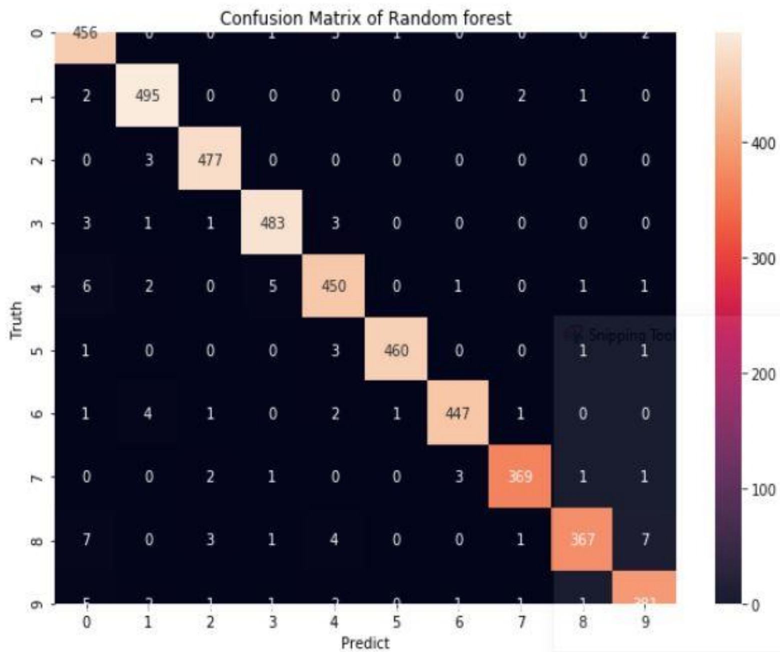
Performance Matrix of KNN

	precision	recall	f1-score	support
0	0.65	0.62	0.63	39
1	0.77	0.92	0.84	48
2	0.62	0.90	0.73	40
3	0.51	0.78	0.62	32
4	0.62	0.60	0.61	43
5	1.00	0.80	0.89	50
6	0.80	0.88	0.84	32
7	0.97	0.70	0.82	54
8	0.89	0.63	0.74	52
9	0.76	0.70	0.73	50
accuracy			0.75	440
macro avg	0.76	0.75	0.74	440
weighted avg	0.78	0.75	0.75	440

Novel Approach

- **Random Forest Classifier**

$$RFfi_i = \frac{\sum_{j \in all\,trees} normfi_{ij}}{T}$$

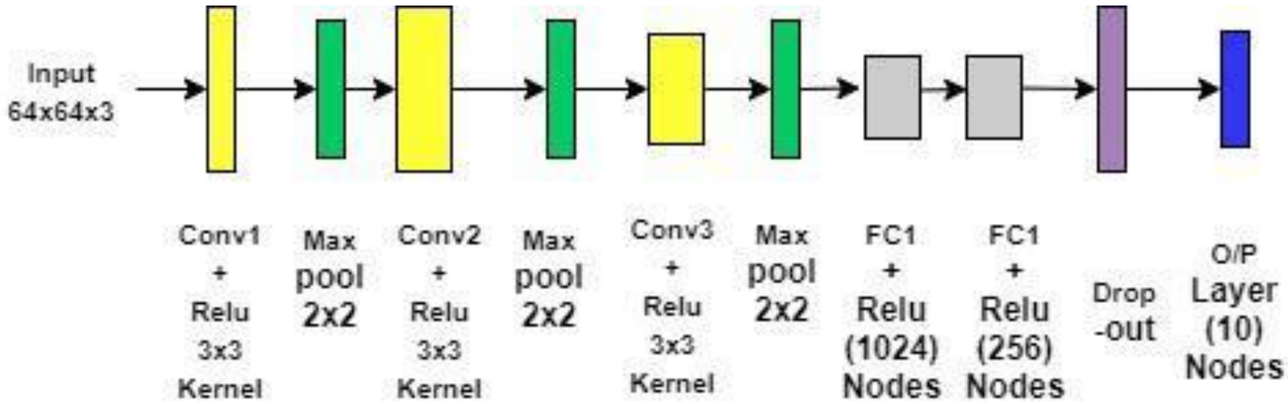


Performance Matrix of Random forest

	precision	recall	f1-score	support
0	0.67	0.85	0.75	463
1	0.82	0.78	0.80	500
2	0.70	0.77	0.73	480
3	0.63	0.79	0.70	491
4	0.71	0.60	0.65	466
5	0.89	0.87	0.88	466
6	0.68	0.84	0.75	457
7	0.95	0.83	0.89	377
8	0.88	0.67	0.76	390
9	0.77	0.64	0.70	395
accuracy			0.76	4485
macro avg	0.77	0.77	0.76	4485
weighted avg	0.77	0.76	0.76	4485

Novel Approach:

- **CNN**



CNN Architecture

```python
classifier = Sequential()
classifier.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = 'relu', input_shape = (240, 240, 3), data_format = 'channels_last'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
classifier.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
classifier.add(Conv2D(filters = 32, kernel_size = (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
classifier.add(Flatten())
classifier.add(Dense(units = 1024, activation = 'relu'))
classifier.add(Dense(units = 256, activation = 'relu'))
classifier.add(Dense(units = 10, activation = 'sigmoid'))
classifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
classifier.summary()
```

Novel Approach:

- **CNN**

```
Epoch 1/10
561/560 [==============================] - 469s 836ms/step - loss: 1.1578 - acc: 0.5941 - va
l_loss: 0.4509 - val_acc: 0.8549
Epoch 2/10
561/560 [==============================] - 405s 723ms/step - loss: 0.3293 - acc: 0.8967 - va
l_loss: 0.2632 - val_acc: 0.9206
Epoch 3/10
561/560 [==============================] - 407s 726ms/step - loss: 0.1832 - acc: 0.9452 - va
l_loss: 0.1786 - val_acc: 0.9431
Epoch 4/10
561/560 [==============================] - 406s 724ms/step - loss: 0.1383 - acc: 0.9564 - va
l_loss: 0.1671 - val_acc: 0.9538
Epoch 5/10
561/560 [==============================] - 404s 720ms/step - loss: 0.1101 - acc: 0.9661 - va
l_loss: 0.1670 - val_acc: 0.9516
Epoch 6/10
561/560 [==============================] - 397s 707ms/step - loss: 0.0952 - acc: 0.9709 - va
l_loss: 0.0925 - val_acc: 0.9721
Epoch 7/10
561/560 [==============================] - 399s 711ms/step - loss: 0.0811 - acc: 0.9758 - va
l_loss: 0.0986 - val_acc: 0.9712
Epoch 8/10
561/560 [==============================] - 403s 718ms/step - loss: 0.0689 - acc: 0.9786 - va
l_loss: 0.1052 - val_acc: 0.9701
Epoch 9/10
561/560 [==============================] - 404s 719ms/step - loss: 0.0634 - acc: 0.9808 - va
l_loss: 0.0829 - val_acc: 0.9761
Epoch 10/10
561/560 [==============================] - 400s 713ms/step - loss: 0.0603 - acc: 0.9805 - va
l_loss: 0.0826 - val_acc: 0.9746
```

Accuracy

Results

- **Existing Models**

Model	Validation Set Accuracy	Validation SetAccuracy with PCA	Validation Set Accuracy with HOG
Linear SVM	71.39	70.33	86.33
Multinomial Naïve Bayes	39.38	NA	45.26
Gaussian Naïve Bayes	56.98	55.11	54.02
Two-layer Neural Network	92.86	92.92	93.12
Softmax Regression	76.81	77.20	78.20

Results

- **Novel Approach**

Model	Validation Set Accuracy
KNN Classifier	74.54
Random Forest	76.13
CNN	97.46

Results: Analysis

- **Artificial Neural Network(ANN)**

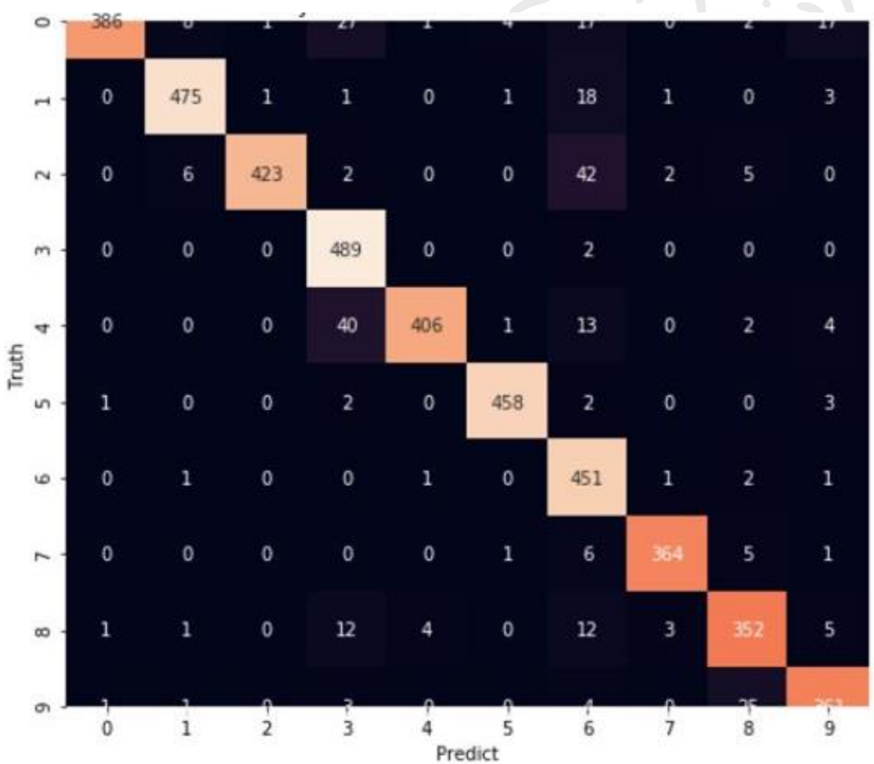
Class	Accuracy(%)
Safe Driving	83.37
Texting-Right	95
Talking on phone-Right	88.12
Texting-Left	99.59
Talking on Phone-Left	87.12
Operating Radio	98.28
Drinking	98.68
Reaching Behind	96.55
Hair and Makeup	90.25
Talking to Passenger	91.39

Class-wise validation set accuracy

Results: Analysis

To more Analysis the performance of our models we also calculated the confusion matrix, Precision, Recall, and f1-score along with Accuracy.

Confusion Matrix



Performance Matrix

	precision	recall	f1-score	support
0	0.99	0.83	0.91	463
1	0.97	0.95	0.96	500
2	1.00	0.88	0.93	480
3	0.85	1.00	0.92	491
4	0.99	0.87	0.92	466
5	0.98	0.98	0.98	466
6	0.80	0.99	0.88	457
7	0.98	0.97	0.97	377
8	0.90	0.90	0.90	390
9	0.91	0.91	0.91	395
accuracy			0.93	4485
macro avg	0.94	0.93	0.93	4485
weighted avg	0.94	0.93	0.93	4485

Results: Comparative Analysis

Existing Models

- 2-layer Neural Network provides best accuracy, among existing classification models.
- On the other hand, Naive Bayes classifier provided the poorest results in terms of accuracy
- In 2-layer neural net, class 8, 9 were predicted with least f1-score(0.92 each) whereas class 5 had highest f1-score(all 0.97)

Novel Approach

- CNN provides best accuracy(97.46%), among the novel approaches implemented.
- KNN classifier proved to be the least accurate classification model.

Conclusions

- After stabilizing the randomness, improving the weight initialization and redoing the hyperparameters tuning of the Softmax regression classifier, the accuracy increased from 28 to 76.81.
- Naïve Bayes is not a good choice for image classification tasks.
- For Softmax classifier and Artificial Neural Net, we could continue tuning other hyperparameters or use some more mature weight initialization techniques like Xavier Initialization or KaiMing Initialization to further optimise the accuracy of prediction.
- As we conclude that CNN is better choice for image classification because CNN is faster than ANN models in terms of computational complexity and give the best result for image classification.

Thank You