Machine Vision CSE480

Final Project Submission

Team Names

Eslam Sayed Rady	1902236
Mohamed Hussein Adel	1802683
Abdelrahman Adel Saeed	1805626

Table of Contents

Project Description

This project aims to explore different techniques used in image classification. The objective is to implement and compare the performance of traditional classifiers and a CNN classifier on RGB images. The project is divided into two milestones, the first milestone focused on Feature extraction and traditional classifiers (KNN, K-means, and SVM) and the second milestone focused on CNN classifier. In Milestone 2, we will use TensorFlow to build a CNN network and train it to classify images. They will also use a pretrained model that is present in TensorFlow and tune it to the classification task at hand. We will then compare the performance of the traditional and CNN classifiers, choosing the classifier with the best performance. The result should contain the performance metrics of each classifier (True Positive, True Negative, False Positive, False Negative) and samples of the classification results. This will give a clear understanding of the performance of each classifier and the best approach for image classification.

The importance of this task

The importance of this task lies in the fact that image classification is a fundamental problem in computer vision and has many practical applications such as object recognition, facial recognition, and medical image analysis. Understanding and comparing different techniques used in image classification can help to improve the accuracy and efficiency of these applications. Additionally, the task also aims to understand the working principles of traditional classifiers and the state-of-the-art CNN based classifiers which is the current trend in image classification. This can help us to understand the current state of the field and the most recent developments in the area.

Milestone (1)

the first milestone focused on Feature extraction and traditional classifiers (KNN, K-means, and SVM)

KNN

- Algorithm and Methods

Figure 1k-nearest neighbors

(k-nearest neighbors) is a non-parametric, instance-based supervised learning algorithm. Given a new observation, it finds the k-number of training examples that are closest to it, and assigns the class label most common among those k-nearest examples. The distance measure used to determine the similarity between the new observation and the training examples can be any valid distance metric (such as Euclidean distance or Manhattan distance). The value of k is a hyperparameter that can be chosen by the user, and is typically chosen through cross-validation. KNN is commonly used for classification and regression tasks.

- Results

```
    return predictions

# Getting the predictions of first 2000 test samples with 3 nearest neighbors
predictions = predict(hog_train_features[0:50000], training_labels, hog_test_features[0:10000], 4)

# Printing the shape of the predictions
print("Shape of the predictions:", predictions.shape)
```

```
distances shape :  (40000, 10000)
INDshape: (4, 10000)
DISshape: (4, 10000)
PredShape: (10000,)
```

```
[ ]
# Calculating the accuracy
Accuracy = calculate_accuracy(training_labels, predictions)

# Printing the accuracy
print("Accuracy: {:.2f}%".format(Accuracy * 100))

```

```
Accuracy: 22.11 %
```

K-means

- Algorithm and Methods

K-means is a clustering algorithm that is used to classify a set of n-dimensional data points into k clusters. The algorithm works by iteratively assigning each data point to the cluster whose mean is closest to it, and then updating the mean of each cluster based on the points that are currently assigned to it.

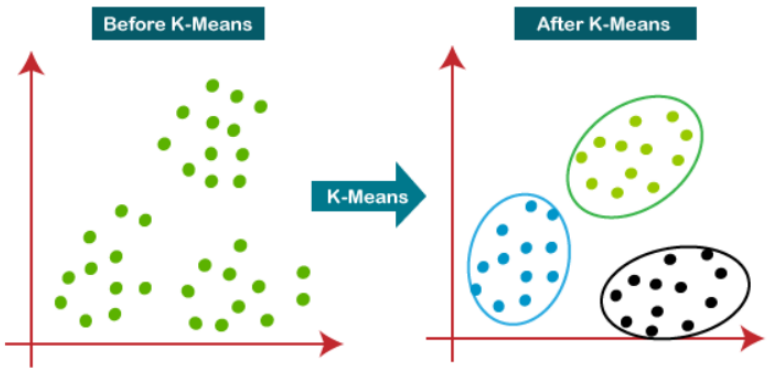


Figure 2K means

The process begins by randomly initializing the means of the k clusters, and then repeating the following two steps until convergence:

Assign each data point to the cluster whose mean is closest to it (using a distance metric such as Euclidean distance).

Update the mean of each cluster to be the mean of the points currently assigned to it.

The end result is k clusters, where each cluster contains a group of similar data points. The number of clusters, k, is a hyperparameter that must be specified by the user in advance. The K-means algorithm is sensitive to the initial choice of cluster means, and multiple runs with different initializations are typically performed to ensure a good solution.

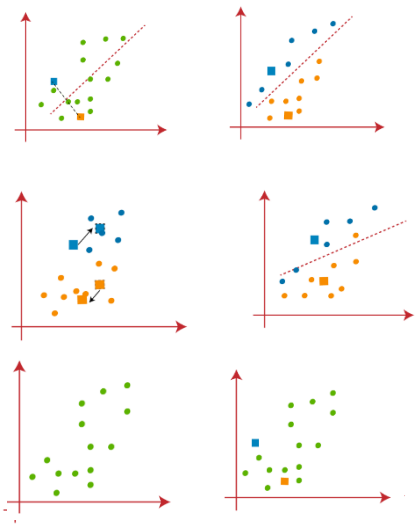


Figure 3K-means

- Results

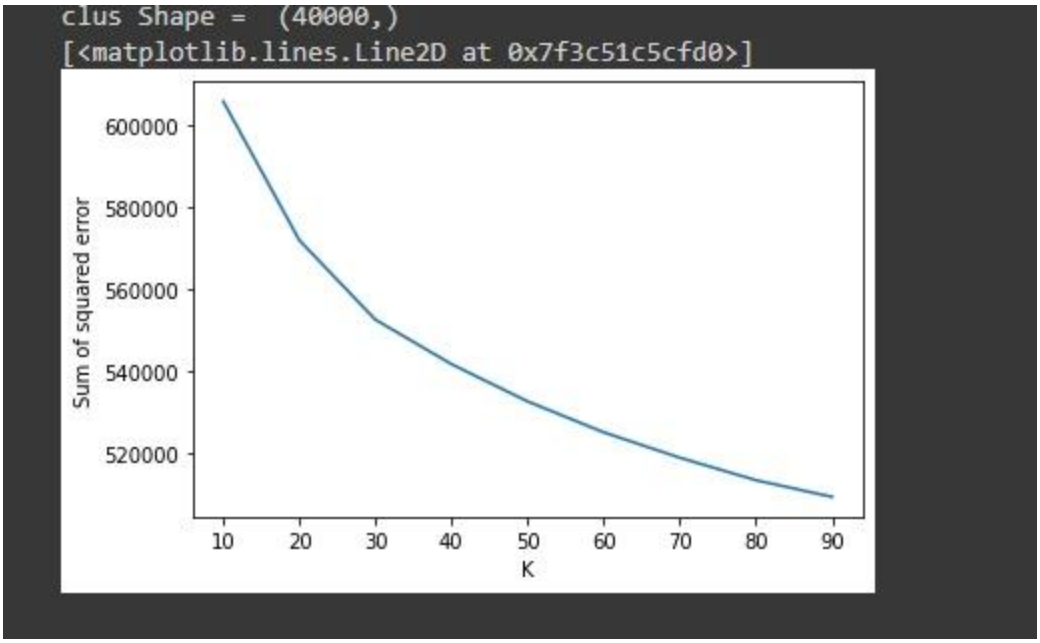
```
# Perform K-Means clustering
clusters, _ = KMeans(hog_train_features[0:20000],100,5)
print("Clusters shape: ", clusters.shape)

# Calculate accuracy
accuracy = accuracy_score(training_labels.flatten()[0:20000], clusters)
print("Accuracy = {:.2f}%".format(accuracy * 100))

```

```
Cen Shape:  (100, 2916)
idx Shape:  (100,)
clus Shape =  (40000,)
IT: 0  E:  898.9993968799497
IT: 1  E:  811.1526776916955
IT: 2  E:  669.3610583509234
IT: 3  E:  658.5920834034375
IT: 4  E:  655.1221828334541
IT: 5  E:  652.6430559025782
IT: 6  E:  651.5855904676114
clusts shape:  (40000,)
Accuracy =  2.2725 %
```

```
clus Shape =  (40000,)
[<matplotlib.lines.Line2D at 0x7f3c51c5cfd0>]
```



- Results

SVM

- Algorithm and Methods

Support Vector Machine (SVM) is a supervised learning algorithm that can be used for classification and regression tasks. The goal of the SVM algorithm is to find the best boundary (or "hyperplane") that separates the data points into different classes.

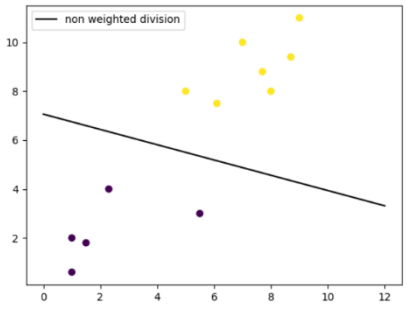


Figure 4linear SVM

In the case of classification, the data points are divided into two classes, and the SVM algorithm finds the hyperplane that maximizes the margin, which is the distance between the hyperplane and the closest data points from each class (called "support vectors"). The intuition behind this is that a larger margin leads to a more robust classifier.

In case of regression, the SVM algorithm finds the hyperplane that minimize the error and maximizes the margin.

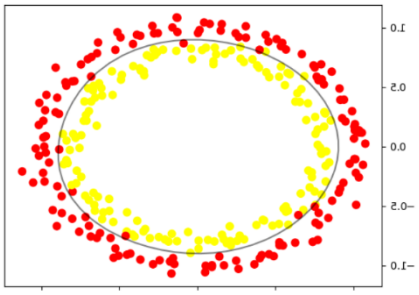


Figure 5non-linear SVM

The SVM algorithm can also handle non-linearly separable data by using a technique called "kernel trick" which maps the input data into a higher dimensional space where a linear boundary can be found.

SVM is particularly useful when the data set has a lot of features, because it's by default able to find the most important features that are needed to classify or predict.

It's a powerful algorithm with a lot of flexibility, but it does require more computational resources than other algorithms and it can be sensitive to the choice of kernel function.

- ## Results

```
[ ]  1 # Create an instance of a radial basis function kernel SVM
     2 svm = SVC(kernel='rbf')
     3
     4 # Fit the model to the training data
     5 svm.fit(hog_train_features, y_train.flatten())
     6
     7 # Predict the labels of the test set
     8 y_pred = svm.predict(hog_train_features)
     9
```

```
1 # Print the confusion matrix
2 print(confusion_matrix(y_test, y_pred))
3
4 # Print the classification report
5 print(classification_report(y_test, y_pred))
6
```

```
[[58  1  0 ...  1  1  4]
 [ 0 36  1 ...  0  0  1]
 [ 2  0  7 ...  2  2  1]
 ...
 [ 0  1  1 ... 17  3  0]
 [ 0  0  2 ...  0 11  1]
 [ 0  0  0 ...  0  0 41]]
              precision    recall  f1-score   support

```

```
          97       0.14      0.17      0.16       100
          98       0.14      0.11      0.12       100
          99       0.39      0.41      0.40       100

    accuracy                           0.39     10000
   macro avg       0.30      0.30      0.29     10000
weighted avg       0.30      0.30      0.29     10000
```

Milestone (2)

the second milestone focused on CNN classifier. In Milestone 2, we will use TensorFlow to build a CNN network and train it to classify images. They will also use a pretrained model that is present in TensorFlow and tune it to the classification task at hand

CNN classifier

- ## Model Description

The model starts with a 2D convolutional layer (Conv2D) with 32 filters, a kernel size of (3, 3), a stride of (2, 2) and a padding of 'same'. The input shape of the data is (32,32,3) which is the width, height, and number of channels of the image. Then it follows by a max pooling layer (MaxPooling2D) with a pool size of (2, 2) and stride of (2, 2).

The next layer is another 2D convolutional layer with 64 filters, a kernel size of (3, 3), a stride of (1, 1), and padding of 'same'. Then it is followed by another 2D convolutional layer with 64 filters, kernel size of (2, 2), stride of (1, 1) and padding of 'same', and a max pooling layer with pool size of (2, 2) and stride of (2, 2).

The next layer is 2D convolutional layer with 128 filters, kernel size of (3, 3), stride of (1, 1) and padding of 'same' and another 2D convolutional layer with 128 filters, kernel size of (3, 3), stride of (1, 1) and padding of 'same' and a max pooling layer with pool size of (2, 2) and stride of (2, 2)

The next layer is 2D convolutional layer with 256 filters, kernel size of (3, 3), stride of (1, 1) and padding of 'same' and a max pooling layer with pool size of (2, 2), stride of (2, 2) and padding of 'same'

Then it flattens the output from the convolutional layers using the Flatten() function, and adds fully connected layers (Dense) with 512, 128, and 64 units respectively. The last layer is the output layer with 100 units and a SoftMax activation function.

The model. Summary() function is used to print a summary of the model architecture, including the number of parameters in each layer.

- ## The parameters :

x: The normalized training data.

y: The one-hot encoded labels for the training data.

shuffle: This parameter is set to True, which shuffles the data before each epoch.

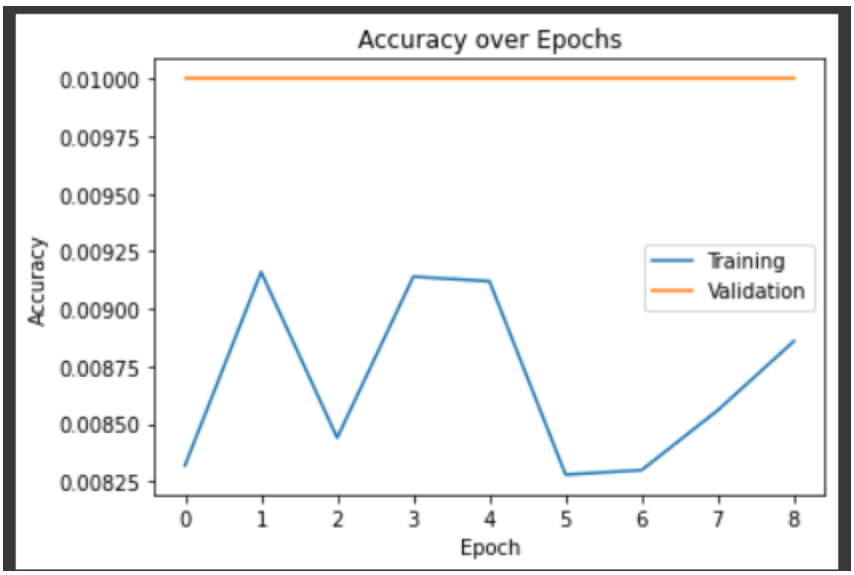
epochs: The number of times the model will cycle through the data.

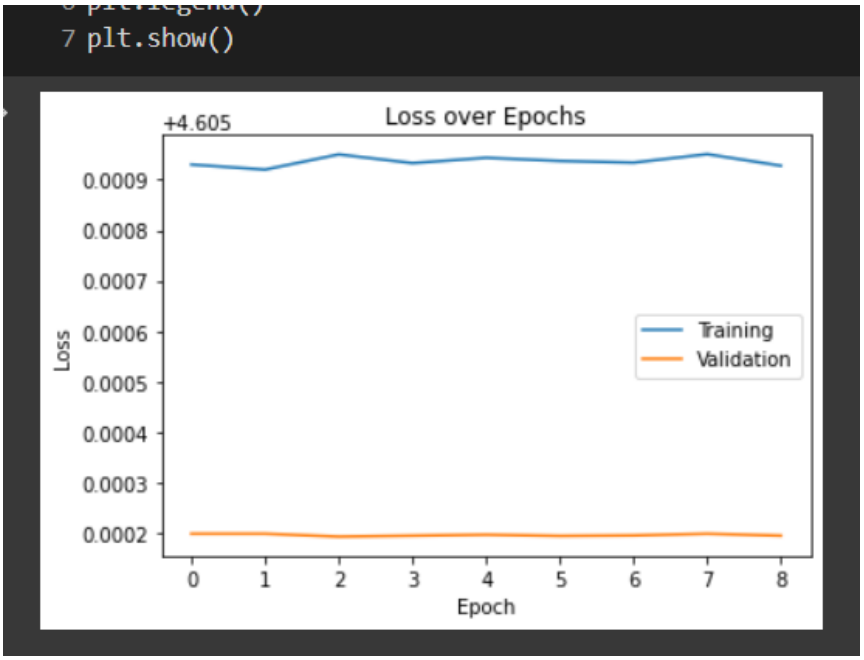
batch size: The number of samples per gradient update.

validation data: The data on which to evaluate the loss and any model metrics at the end of each epoch. The model will not be trained on this data.

callbacks : is an Early Stopping callbacks which stop the training if the performance is not improving over 'patience' number of epochs.

- ## Results of M1





- ## Model2 Description

The model is initialized as a Sequential model, which means that the layers will be added in a linear stack one after the other.

The first layer added is a convolutional layer using the Conv2D function, which takes in several arguments: 32: number of filters

kernel_size: the size of the filters, in this case (3, 3)

strides: the step size for moving the filters across the input image, in this case (2, 2)

padding: the type of padding to use around the input image, in this case 'same' which means the output will have the same dimensions as the input

activation: the activation function to use, in this case 'relu'

input_shape: the shape of the input image, in this case (32, 32, 3)

Then, the first max pooling layer is added using the MaxPooling2D function. This layer takes the maximum value from the previous layer in a defined window size (2, 2) and strides (2, 2)

Then, a dropout layer is added to prevent overfitting.

The next part of the code adds two more convolutional layers, which are very similar to the first one, but with different filter numbers (64).

Then, the second max pooling layer is added in a similar way to the first one.

Then, a GlobalAveragePooling2D layer is added to reduce the dimensions of the output from (None, 4, 4, 100) to (None, 100) and make it compatible with the final dense layer.

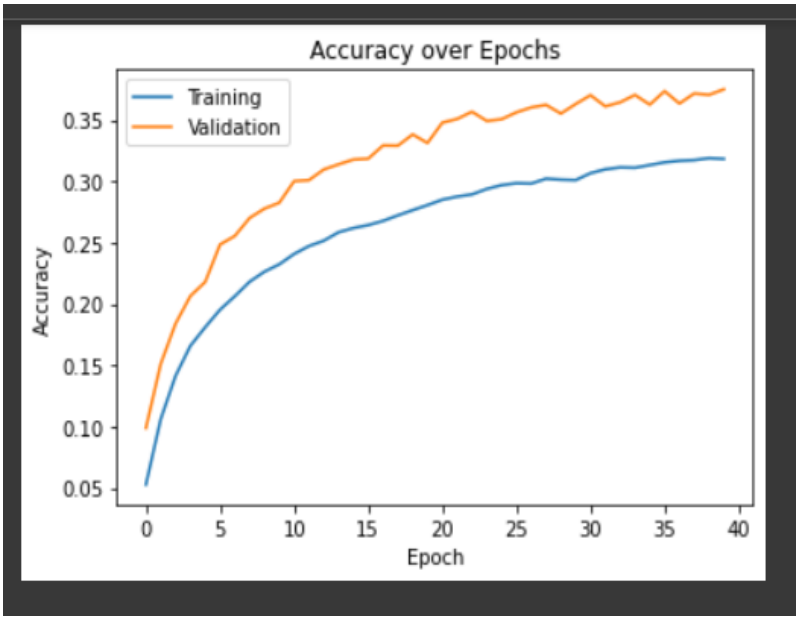
Then, the final dense layer is added with 100 neurons and a softmax activation function, this is the output layer where the predictions will be made.

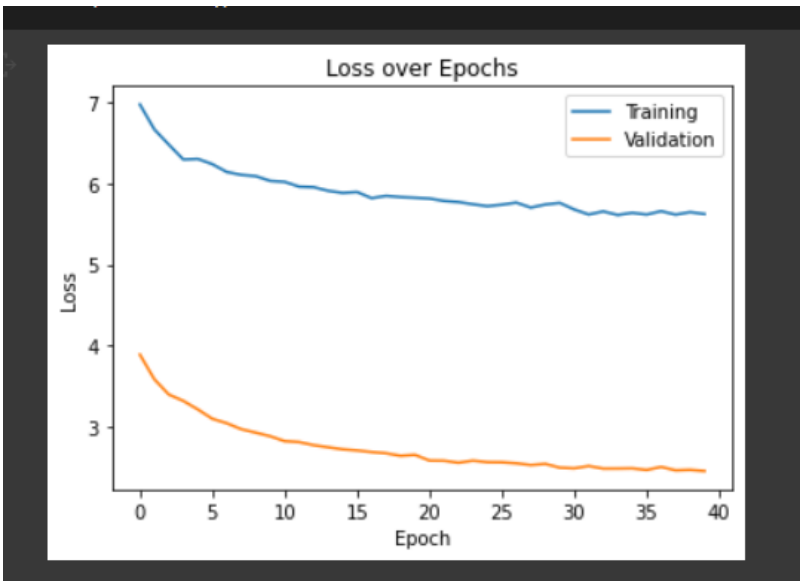
Finally, another dropout layer is added to prevent overfitting.

This model has multiple convolutional layers, max pooling layers, dropout layers and a dense layer, that allows the model to learn features from the image while reducing the dimensions of the data, then make the predictions.

- Results of M2

```
Epoch 36/40
1563/1563 [==============================] - 43s 28ms/step - loss: 5.6166 - accuracy: 0.3155 - val_loss: 2.4739 - val_accuracy: 0.3736
Epoch 37/40
1563/1563 [==============================] - 43s 27ms/step - loss: 5.6579 - accuracy: 0.3168 - val_loss: 2.5116 - val_accuracy: 0.3635
Epoch 38/40
1563/1563 [==============================] - 45s 29ms/step - loss: 5.6155 - accuracy: 0.3174 - val_loss: 2.4694 - val_accuracy: 0.3717
Epoch 39/40
1563/1563 [==============================] - 44s 28ms/step - loss: 5.6460 - accuracy: 0.3190 - val_loss: 2.4752 - val_accuracy: 0.3706
Epoch 40/40
1563/1563 [==============================] - 44s 28ms/step - loss: 5.6232 - accuracy: 0.3184 - val_loss: 2.4610 - val_accuracy: 0.3751
```

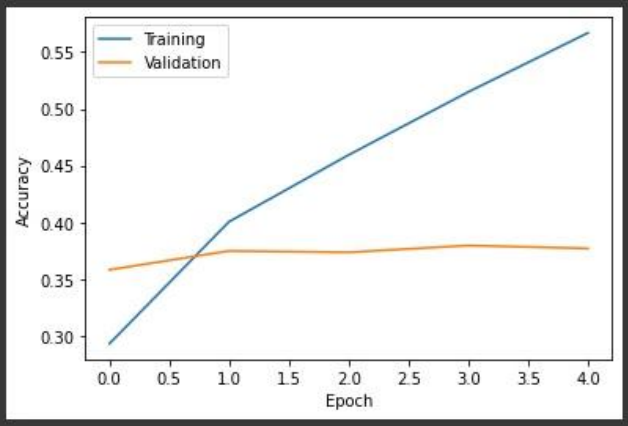




DenseNet201 Model

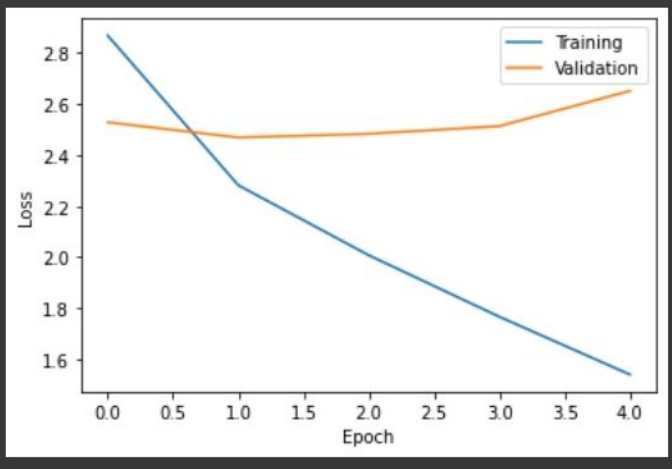
```
# Plot the training and validation accuracy over the number of epochs
plt.plot(result.history['accuracy'], label='Training')
plt.plot(result.history['val_accuracy'], label='Validation')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

```




```
# Plot the training and validation loss over the number of epochs
plt.plot(result.history['loss'], label='Training')
plt.plot(result.history['val_loss'], label='Validation')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

```



Comment

Traditional classifiers such as Support Vector Machines (SVMs), Random Forest, and K-Nearest Neighbors (KNNs) are based on hand-engineered features and have been widely used in many applications. They are simple to understand and implement, and they can handle small to medium-sized datasets efficiently. However, they may not perform as well on complex and large datasets, particularly when the data has a high dimensionality.

Convolutional Neural Networks (CNNs), on the other hand, are a type of deep learning classifier that is well-suited for image and video recognition tasks. CNNs automatically learn features from the data and are able to handle large and complex datasets. They are able to capture spatial hierarchies and local patterns in images, which makes them well-suited for image classification tasks. CNNs can also generalize well to unseen data, and they can achieve state-of-the-art results on many benchmark datasets.

In general, CNNs tend to perform better than traditional classifiers on image and video recognition tasks, while traditional classifiers may perform better on simpler datasets or tasks where interpretability is important.

Appendix

- Milestone 1:

  https://colab.research.google.com/drive/1jvf-Cdudh3KI4q5aeKtI3-jAC_FtErhq#scrollTo=MkV7xGiPct3i

- Milestone 2.1:

  https://colab.research.google.com/drive/1Bn8Ij3Lu5JSqd3c1KWFWEKY8bTJXf_UE#scrollTo=EdHoJLEFSIrh

- Milestone 2.2:

  https://colab.research.google.com/drive/1Vw0lEyCka780Q1mqIrWL1sTK1WvvDsq0#scrollTo=FgbUDb4GmasN

References:

Beyer, Kevin; et al. "When is "nearest neighbor" meaningful?"

Ng, Andrew Y. (2012). "Learning feature representations with *k*-means"

Chang and Lin, LIBSVM: A Library for Support Vector Machines.