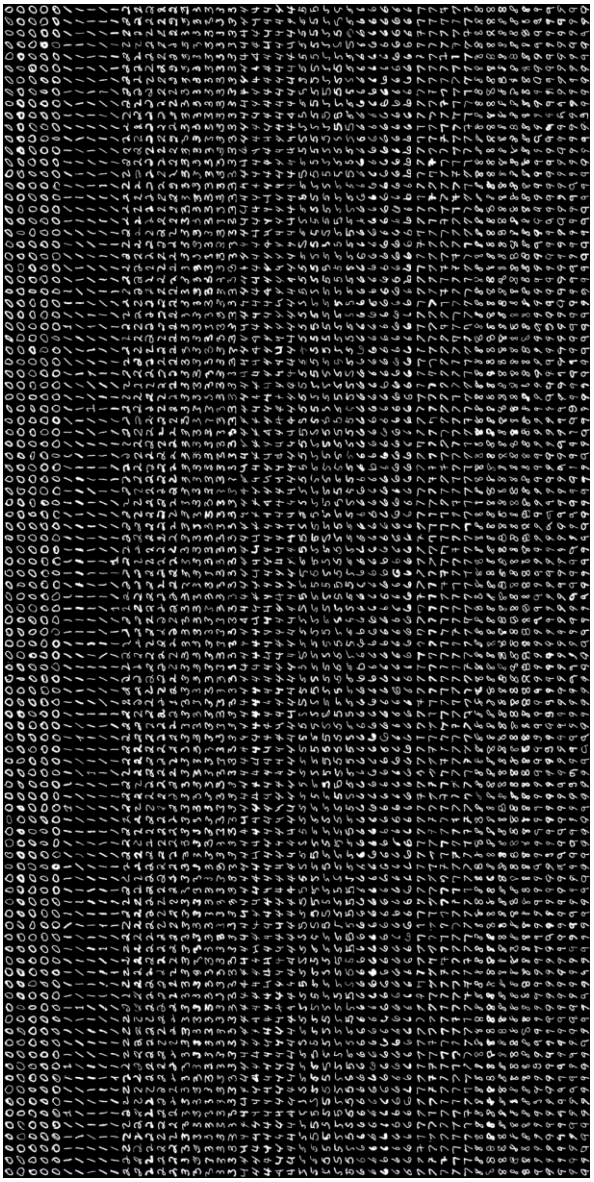


IMAGE CLASSIFICATION

ABSTRACT

This study explores a computer vision problem by using machine learning techniques for image classification and evaluate models and their accuracies.

Jevi Waugh – Model Analysis

Table of Contents

1 Introduction

The primary objective of this research is to explore and implement certain machine learning techniques and algorithms for a computer vision problem. The models and techniques ranges from K-NN nearest neighbour, linear classifiers such as support vector machines, SIFT and pre-trained convolution neural networks. This research paper endeavours to produce models that will work on image classification and produce predictions from complex sequential processes. The classification will refer from a dataset consisting of tiny digit images ranging from 0-10 in a sequential format.

2 Scope and methodology

The scope of this research is structured into 5 different tasks that guides the process of different models and their selection of hyper-parameters, training, and classification evaluations.

2.1 Task 1: Data Preparation

1. **Image processing and Data extraction:** Converts the image to grayscale and utilised basic computer vision techniques to segregate digits from the image.
2. **Data storage:** Partitioned the extracted data into 80% for training and 20% for testing.

2.2 Task 2: K Nearest Neighbour for Image classification

1. **Model Training:** Implementation of the K-NN algorithm.
2. **Hyper-parameter tuning:** Deploying K-NN with multiple k-values and comparison.
3. **Accuracy and classification:** Prediction of standardised accuracy and other metrics.
4. **Confusion matrices and evaluation:** The model's performance and accuracy are analysed thoroughly through confusion matrices and other forms of classification evaluation techniques.

2.3 Task 3: Support vector machine for image classification

1. **Multi-class model training and hyper-plane formula:** Support vector multi-class training
2. **Fine-tuning with Termination Criteria and Cost parameter:** Fine tuning with parameters for higher accuracy
3. **Hyper-plane and margin analysis:** Analytical elucidation of the hyperplane fitting apart the binary classes.
4. **Confusion Matrix:** Confusion Matrix results
5. **Model comparison with K-NN technique:** Comparison between K-NN and linear classifier

2.4 Task 4: Bag of visual words

1. **Perception Pipeline:** The Perception pipeline for Bag of visual words (BoVW).
2. **Feature extraction:** Extraction of sift features class by class for training and testing separately and BoVW.
3. **K-means:** Extracting Visual words and experimentation of K values.
4. **Histogram and 1-NN training:** Trained classifiers based off histogram features.
5. **Accuracy and performance metrics:** Model evaluation through performance metrics and accuracies.

2.5 Task 5: Convolutional neural networks

1. **Models and validation data:** Utilising 3 pre-trained CNN architectures (AlexNet, VGG and ResNet).
2. **Fine-tuning:** Adjusting Learning rate, batch size and slicing the network's layers.
3. **Regularisation:** utilizing techniques such as data augmentation.
4. **Hyper-parameter tuning:** Conducting different iterations (epoch) under different learning rates and maximum testing data.
5. **Performance Metrics:** The accuracy of the model class-wise and model wise.
6. **Model evaluation and comparison**: The evaluation and comparison of CNN models

3 Significance and innovation

As computer vision perpetuates this culture of eccentric image understanding and 3d scene reconstruction, it continues to be a vital component in several applications such as self-driving cars, robotics, space exploration, biology, and the ability to interpret and classify images remains a keystone in the field of computer vision. This research paper provides practical insights for computer vision tasks by using efficient machine learning techniques. The aim of this introduction is to lay out the framework for the profoundly debate and analysis that will follow.

4 Task 1: Data Preparation

4.1 Image processing

First off, the image (digits.png) was converted to a grayscale image due to its computational efficiency as opposed to 3 channels in colour images. This simple conversion makes it a practical choice in computer vision task as it saves storage and speeds up processing time and has enough information to work with classification models.

4.2 Data Extraction and partitioning

The raw pixels were extracted and were interpreted as digit due to its 20 by 20 format. The image consisted of digits ranging from 0-9 with 5 rows each of every number and each row consisted of 100 digits each. In summary, there were 500 digits per class and 5000 altogether. Furthermore, for every row and for every 10 digits, the data were extracted and stored into training and testing arrays of 20 by 20 feature vector. Every 4 rows of a specific class were allocated to the training array and the last one to the testing array maintaining the 80% and 20% split dataset rule. The following figure shows the feature extraction implementation.

```
all_digit_numbers = np.empty((0,20,20),dtype=np.uint8)
training_digits = np.empty((0,20,20),dtype=np.uint8)
test_digits = np.empty((0,20,20),dtype=np.uint8)

for k in range(10):  # 10 Digits
    for r in range(5):  # 5 rows
        # Train every 4 rows and test the last rows
        # It works but it's redundant
        for i in range(100):  # 100 numbers
            digit = digit_image[(k*5+r)*20:(k*5+r+1)*20, i*20:(i+1)*20]
            digit = np.expand_dims(digit, axis=0) # Adding another dimension due to the array's initial size
            all_digit_numbers = np.append(all_digit_numbers, digit, axis=0) # Just in case we need to the full size
            if r < 4:
                # print("Appending 80%")
                training_digits = np.append(training_digits, digit, axis=0)# Append 4 rows for training
            else:
                # print("Appending 20%")
                test_digits = np.append(test_digits, digit, axis=0)  # Append the last row for testing
```

4.3 Storage and labelling

The data were stored in specific folders such as “Train” and “Test” and every digit has a unique filename for image classification uses and labelling was an important step for that method. The filename was in the form of **"digit_{i}_row{j+1}_id_{counter}.jpg"**. So every digit’s filename started with its actual digit and its corresponding row and also its unique id due to its variant digit representation making the image different from other digits storage-wise. This makes the distinction of classes within folders and can be easily picked up when being trained by a CNN model to recognise the class label. The digits were also saved as jpeg since it uses lossy compression which allows the images to be stored at a smaller file size and maximises the space. It uses 10:1 compression ratio which is beneficial when dealing with large dataset as it reduces the overall disk space required and speeds up I/O operations. The figure below provides an implementation with its allocation to training and testing folders.

```
training_elements = 0
testing_elements = 0

for i in range(10): # 0-9 digits
    counter = 1
    for j in range(4): # 4 rows
        for k in range(100):
                    # row1 = 0-100 # row2 = 100-200 # row3 = 200-300# row4 = 300-400
            filename = f"digit_{i}_row{j+1}_id_{counter}.jpg"
            # 0-3999 == 4000 elements which is 80 percent for training
            # print(filename)
            training_folder_path_1 = os.path.join(training_folder_path, filename)
            if os.path.isfile(training_folder_path_1) == False:
                cv.imwrite(training_folder_path_1, np.uint8(training_digits[training_elements])) # Saving the image in the Train folder 0 till 4000
            counter += 1
            training_elements += 1 # till 4000 elements

    counter = 1
    for l in range(100): # There's only 1000 elements for testing
        filename = f"digit_{i}_id_{counter}.jpg"
        testing_folder_path_1 = os.path.join(testing_folder_path, filename)
        if os.path.isfile(testing_folder_path_1) == False:
            cv.imwrite(testing_folder_path_1, np.uint8(test_digits[testing_elements])) # Saving the image in the Test folder 0 till 2000
        counter += 1
        testing_elements += 1
```

4.4 Normalising and vectors

After successful extraction of the data, the dimensions of the extracted images were reshaped from a 20 by 20 patch to a row vector of 400 columns that facilitates efficient matrix operations that are exclusive to machine learning techniques. The datatype was readjusted to float32 as this format uses less memory and has improved computational

speed. This normalisation and standard simplification are assured to streamline data and ensuring interoperability with machine learning models as shown below.

```
# At the moment the images are 20 x 20
# Convert it to a row vector of 400
train_array = np.array(training_images).reshape(-1,400).astype(np.float32)
test_array = np.array(testing_images).reshape(-1,400).astype(np.float32)
```

5 Task 2: K Nearest Neighbour for Image classification

To commence with K-NN implementation, the labels for the dataset were created and were transformed to a row vector for compatibility with the dataset since the primary features were originally flattened to a 400-row vector since there were 400 features in a 20 by 20 image patch. The implementation is shown below.

```
c = np.arange(10)  ## create an array from 0-9
train_labels = np.repeat(c,400)[:,np.newaxis] # 400
test_labels = np.repeat(c,100)[:,np.newaxis]# 100 is
# The [:,np.newaxis] turn it into a 1 row vector
```

5.1 Model Training and Hyper-parameter tuning

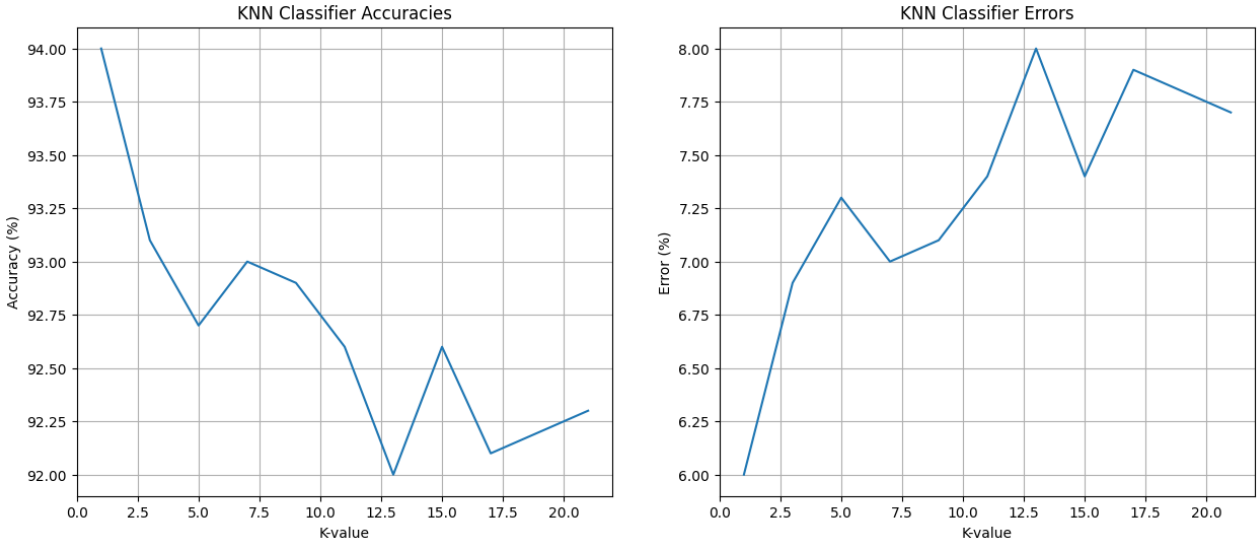
Implementation of K-NN was done throughout multiple values of K to determine the optimal hyper parameter and it was one of the most critical decisions of which has full control over the dictatorship of nearest neighbours that the K-NN technique considers when making the classification. In the image below the neighbourhood size k ranging from 1 to 22 incremented by 2, specifically selecting odd values because K-NN assigns a class based on majority vote among the neighbours and ties are avoided by selecting odd numbers only. Odd number of neighbours guarantees a clear majority vote, strengthening the model's conclusion.

```
knn = cv.ml.KNearest_create()
knn.train(train_array, cv.ml.ROW_SAMPLE, train_labels)

k_values = []
for i in range(1,22, 2):
    k_values.append(i)
 # odd values to avoid ties due to the nature of even numbers
accuracy_list = []
for i, k_value in enumerate(k_values):
    start_time = time.time()
    ret,result,neighbours,dist = knn.findNearest(test_array, k=k_value)
    end_time = time.time()
    elapsed_time = end_time - start_time
    matches = result == test_labels
    correct = np.count_nonzero(matches)
    accuracy = correct*100.0/result.size
    accuracy_list.append(accuracy)
    # Find the error rate for k
    error_rate = 100 - accuracy_list[i]
```

5.2 Hyper-parameter tuning (k)

Within the deployment of the K-NN algorithm, it had seemed that the optimal value of the neighbourhood size k has a prominent impact upon the model's performance and classification accuracy. For example, the algorithm achieves a high accuracy score of 94% with an error rate of 6.00% when k is 1. K-NN was experimented with multiple odd values and the only k value that had the lowest accuracy was 13 with a 92% accuracy with an error rate of 8%. Although the drop is not significant from the k value 1, it's still significant to mention some closer values have gone higher than the previous value, however not higher than 94% and it's always about 0.5% higher as shown in the graph below presents the accuracies and error rates for different neighbourhood values.



It's critical to understand that even though the optimal neighbourhood value 1 offers a high standardised accuracy and reasonable computational efficiency, it must still be noted that if it's not monitored, the model may be prone to overfitting and this is where the model fails predict on the test data, if the number of features compared to the number of classes is significantly bigger and it learns the noise from the dataset. Which is why it is important to keep tuning this hyper-parameter and if *k = 1* overfits in the future, a closer k values that produces reasonable results may be selected. Although currently an accuracy of 94% for the dataset is generally very good upon which will produce reasonable image classification results.

5.3 Accuracy and classification

An accuracy of how well the classifier is correct was assessed, examining the performance metrics for individual classes shows how accurate it was in certain cases and how incorrect it was in other. For this scenario, three classes 0, 6 and class 9 will be analysed closely.

From the data below, class 0 and 6 have an exact outstanding accuracy of 100% primarily due to its high true positive (TP) rate of 100 out of 100 samples. The following formula provide the accuracy of which indicated how the model would get the classification correct most of the time which is crucial for the dataset.

Standard Accuracy = $\frac{TP+TN}{TP+FP+TN+FN}$

Moreover, the precision $\frac{TP}{TP+FP}$ for class 0 and class 6 are 0.952 having a high degree of exactitude in the positive classifications of the model. Additionally, recall $\frac{TP}{TP+FN}$ depicts that K-NN captures all the actual positive class 0 and class 6 digits. Finally, the F1 score $2 * \frac{PRECISION*RECALL}{PRECISION+RECALL}$, a mean of Precision and Recall is at 0.961. Although this is great results, it's worth noting that the model produces a slight error rate of 0.5% with a false positive (FP) rate of 5 for both classes.

Class 9 has an exceptional accuracy of 98.1% but being the lowest out of all the classes with a true positive rate of 93 from 100 samples. It has a precision of 0.886 with a false positive of 12 leading to a lower precision and shows that seven non 9 digits had been misclassified as 9. Moreover, it has a recall of 0.93 which shows that it has a high accuracy for positive cases that are identified correctly. Finally, class 9 has an F1 score of 0.907 and an error rate of 1.9% which is the highest out of all the classes. An elucidation of how the nines are gotten wrong are explained later on with a visual plot.

KNN PERFORMANCE METRICS

Class	TP	FN	FP	TN	STANDARD ACCURACY	PRECISION	RECALL	F1 SCORE	ERROR RATE
0	100	0	5	895	99.5%	0.952	1.0	0.975	0.5%
1	97	3	4	896	99.3%	0.96	0.97	0.965	0.7%
2	87	13	2	898	98.5%	0.978	0.87	0.921	1.5%
3	89	11	7	893	98.2%	0.927	0.89	0.908	1.8%
4	96	4	8	892	98.8%	0.923	0.96	0.941	1.2%
5	92	8	7	893	98.5%	0.929	0.92	0.924	1.5%
6	100	0	5	895	99.5%	0.952	1.0	0.975	0.5%
7	97	3	4	896	99.3%	0.96	0.97	0.965	0.7%
8	89	11	6	894	98.3%	0.937	0.89	0.913	1.7%
9	93	7	12	888	98.1%	0.886	0.93	0.907	1.9%

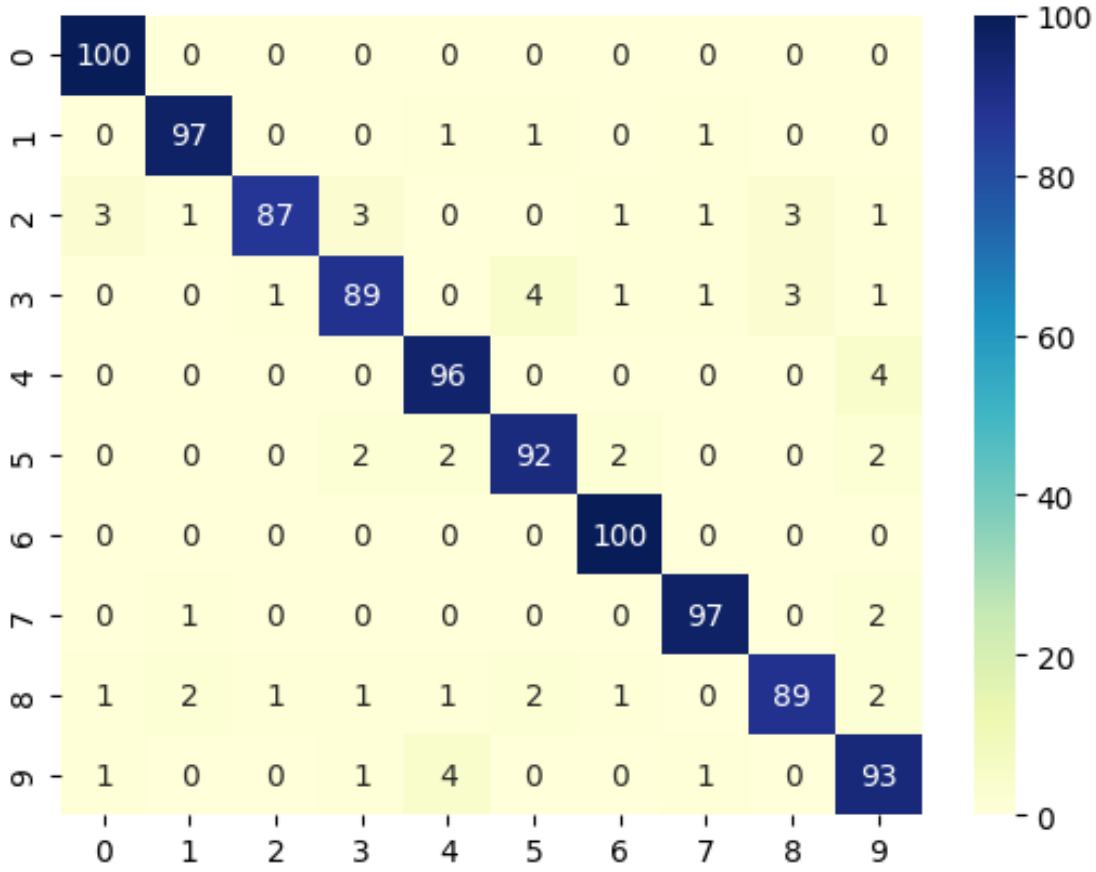
While the performance of K-NN shows outstanding results, some classes suffer from precision which indicates a higher rate of false positives in the prediction and other have a lower recall that missed the true positive values and, in this case, certain classes such as class 3 and 8 have 11 false negative which are is a substantial amount missed. However, given that we had an 80% training dataset and only 20% of the data is being tested, this ensures for more confidence within the validity of performance metrics and making the results more generalised to new unseen data.

5.4 Confusion Matrix and evaluation

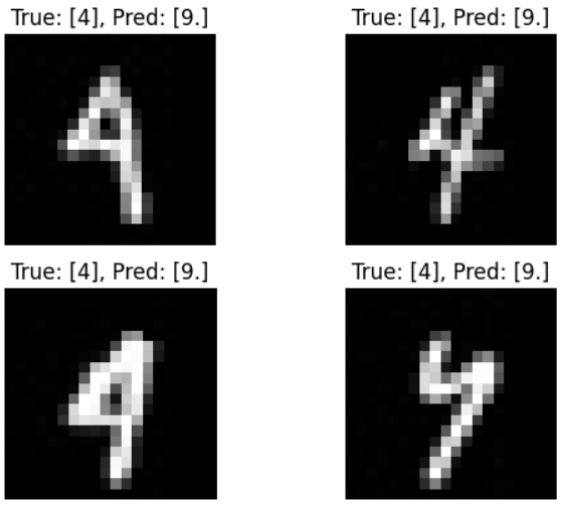
The model's performance and accuracy are thoroughly examined using confusion matrices, and 7 non 9 digits are analysed to provide a detailed explanation as to why it has an incorrect prediction. The following confusion matrix shows the True positive along the diagonal line of the figure and some incorrect values along the plot itself.

As it can be seen in the confusion matrix below, all classes are above 85% accuracy while most resulting of 7 classes are above 90% which has a great significance upon the recognition of digits compared to others written in multiple style considering the dexterity of

human hands can be versatile with digit variants. Class 1 has 97 digits accurate but has had 4 and 5 confused with class 1 considering that if 4 and 5 were written very thin and long, the model could have interpreted it as class 1. This is also a similar case for the rest of the misclassified numbers.



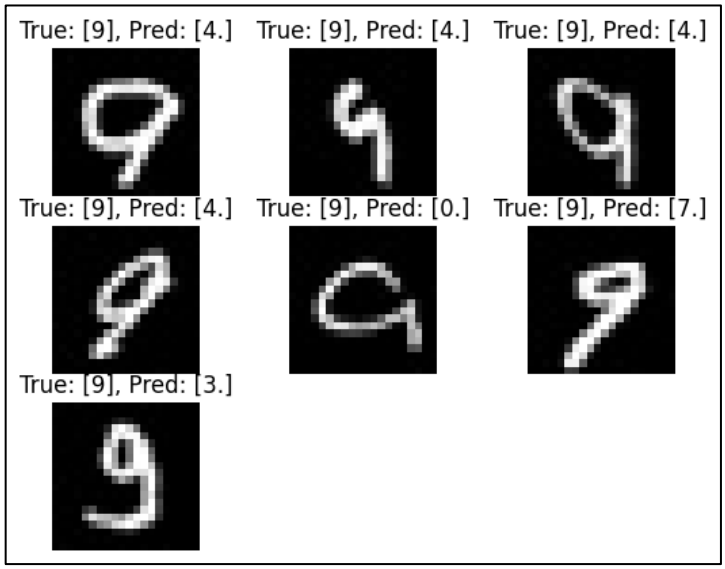
5.5 Misclassification of Class 4



By analysing the image below, it seems that class 4 classified four numbers as nines considering that the variant images share visual features with 9's which have similar curve and angles specially at the top of the digit where the head of 4 is formulated, it looks like the head of 9 in the images and thus resulting in the misclassification of 9. This problem could also have been triggered by scaling factors within the dataset that possibly could not have been trained for this specific use case and the features extracted were not sufficiently enough to distinguish between 4's and 9's.

5.6 Misclassification of Class 9

Since the confusion of class 4 and 9 have already been discussed and the features that are being conflated with each other, the focus is now upon 3 misclassified 9 digits. In the image shown below it seems that the digit that has been classified as 0 has a small head with a short tail which is a very bizarre way of writing 9 but it is still the number 9 because it was trained as 9. It is also one of those things where if someone wrote it as a 0 instead of 9, it would have been trained as 0 and not 9 and the model would have been trained on it and believed it. So, interpretation and the correct dataset also have an impact for this recognition task because it is difficult to predict a variant of a digit of which the writer could have meant another digit by ignoring the primary features of class 9 such as the head as a zero with curved tail or long tail as primarily written by people. In this case it looks like a zero with a short tail and the probability that the model considers this digit as 0 to 9 is high due to the short tail and big head that mostly matches the primary features of the digit 0. The next one has been predicted as 7 dues to its thin nature and diagonal look of the digit of which 7 is mostly captured. It could also be seen that the head of 9 in that image is very much oval and almost would be on a flat line if there were a few pixels closer. The last misclassified one is predicted as 3 because the typical digit 3 has 3 curved horizontal layers within the digit and the curved tail of the number 9 in the image below is very much flat of which may have conflated it to the digit 3. Moreover, these are the reasons why class 9 had the least accuracy out of all classes leading to a biased prediction of class 9 due to scaled features within the testing class.



6 Task 3: Support vector machine for image classification

6.1 Multi-class model training and hyper-plane formula

A traditional support vector machine is typically trained through a binary implementation so that the hyperplane maximises the support vectors between two distinct categories and in this scenario with the image classification from our digit dataset, the classes have been divided into 1 and -1. Given the implementation below, a support vector machine has been created for every class. For example, if class 0 is being tested the digits 1-9 will be classified as one other class (-1) for this binary implementation because if the nature of support vector machines. Hence this one VS the rest method was used where multiple SVM classifiers are trained discriminating one class from all the remaining classes. Moreover, since the hyperplane formula x : $f(x) = \beta^T x + \beta_0$ was not directly able to be manipulated for

experimentation, where the weight vector $\beta^T x$ could have been multiplied by a scaler to change its magnitude and its strength of the vector determines the distance from the hyperplane to the support vectors and the idea is to maximise the hyperplane to have a better generalisation of the distinct classes. The bias vector β_0 could also be changed to move the hyperplane upwards or downwards. This experimentation could change the position of the hyperplane while affecting the classification results. But because OpenCV's implementation of SVM was utilised, it is generally the case that the formula isn't altered after training, which is why the hyper-parameters such as the termination criteria, number of iterations and epsilon were altered beforehand a few times to get the most optimal accuracies across all SVM classifiers.

```
# initially they are lists for simplicity sake
binary_train_labels = [1 if label == i else -1 for label in train_labels]
binary_test_labels = [1 if label == i else -1 for label in test_labels]

# Convert the lists to arrays for futher analysis and use
binary_train_labels, binary_test_labels  = np.array(binary_train_labels), np.array(binary_test_labels)

svm = cv.ml.SVM_create()
svm.setType(cv.ml.SVM_C_SVC)
svm.setKernel(cv.ml.SVM_LINEAR)
svm.setC(0.000001)  # Set the cost parameter to 0.1
```

6.2 Fine-tuning with Termination Criteria and Cost parameter

Originally, the parameters were set to 100 iterations with an epsilon of 1e-2 which of gave SVM a lot of time to find an optimal hyperplane however because 1e-2 converges faster but at the expense of certain misclassifications and accuracies. However it does has an empecable accuricy of 99.8% for class 1 which is only 0.2% less than a 100%. The image below shows the class-wise accuracy with the currrent paramters set.

```
Binary clasification for Multi-class classification

The accuracy for the class 0 vs the rest is: 98.7% with an error rate of 1.30%
The accuracy for the class 1 vs the rest is: 99.8% with an error rate of 0.20%
The accuracy for the class 2 vs the rest is: 95.2% with an error rate of 4.80%
The accuracy for the class 3 vs the rest is: 94.5% with an error rate of 5.50%
The accuracy for the class 4 vs the rest is: 96.7% with an error rate of 3.30%
The accuracy for the class 5 vs the rest is: 91.6% with an error rate of 8.40%
The accuracy for the class 6 vs the rest is: 98.1% with an error rate of 1.90%
The accuracy for the class 7 vs the rest is: 96.5% with an error rate of 3.50%
The accuracy for the class 8 vs the rest is: 90.4% with an error rate of 9.60%
The accuracy for the class 9 vs the rest is: 92.2% with an error rate of 7.80%

The highest One-Vs-Rest accuracy was 99.8% for class 1 with an average accuracy of 95.37.
```

Throughout multiple experimentations, the termination criteria have a significant impact upon the accuracy of certain classes and its average accuracy across all classes changes slightly. The models were fined tuned with maximum iterations of 1000 at first with an epsilon of 1e-8. As shown in the performance metrics figure below, the model has an average accuracy of 99.70% of all classes which ultimately connotes that all classes are performing at this level on an average whereas the highest one vs the rest accuracy was 99.7 for class 1. This termination criteria with the number of iterations have had a lot of time to adjust and learn from it mistakes by also having a stricter convergence criterion of 1e-8 epsilon which ultimately allows the refined hyperplane to fit, therefore resulting in a high

average accuracy of 97.70%. The algorithm will keep iterating until the change in the objective function between the iterations is small which is also very computationally expensive however, it ensures a higher accuracy for every class even though it converges slower. So here it seems that for a higher accuracy across all classes with the previous accuracy being 95.37% and this new accuracy of 97.70% with more iterations and a different epsilon value rages a difference of 2.33% higher than before across all classes making the algorithm better at maximising the hyperplane where close variant classes resembling to other classes are not classified as other but instead to its true value. However, class 1 had a drop of 0.1% which is insignificant compared to the progress of other multi-classification.

Another parameter that was fine-tuned was the cost parameter of which essentially influences the trade-off between the minimisation of classification errors and maximising the hyper-plane margin. Setting it to a low value such as 0. 000001 technically makes the model focus more on maximising the hyperplane at the expense of certain errors, however this value was found to be the one that produces less errors and fits the training data better.

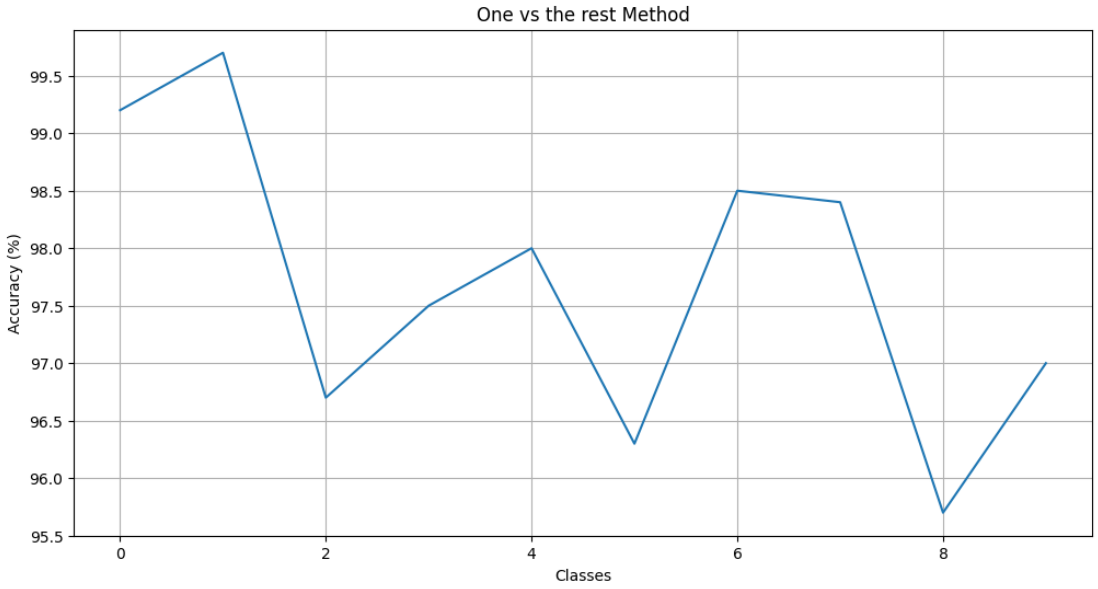
```
Binary clasification for Multi-class classification

The accuracy for the class 0 vs the rest is: 99.2% with an error rate of 0.80%
The accuracy for the class 1 vs the rest is: 99.7% with an error rate of 0.30%
The accuracy for the class 2 vs the rest is: 96.7% with an error rate of 3.30%
The accuracy for the class 3 vs the rest is: 97.5% with an error rate of 2.50%
The accuracy for the class 4 vs the rest is: 98.0% with an error rate of 2.00%
The accuracy for the class 5 vs the rest is: 96.3% with an error rate of 3.70%
The accuracy for the class 6 vs the rest is: 98.5% with an error rate of 1.50%
The accuracy for the class 7 vs the rest is: 98.4% with an error rate of 1.60%
The accuracy for the class 8 vs the rest is: 95.7% with an error rate of 4.30%
The accuracy for the class 9 vs the rest is: 97.0% with an error rate of 3.00%

The highest One-Vs-Rest accuracy was 99.7% for class 1 with an average accuracy of 97.70.
```

6.3 Hyper-plane and margin analysis

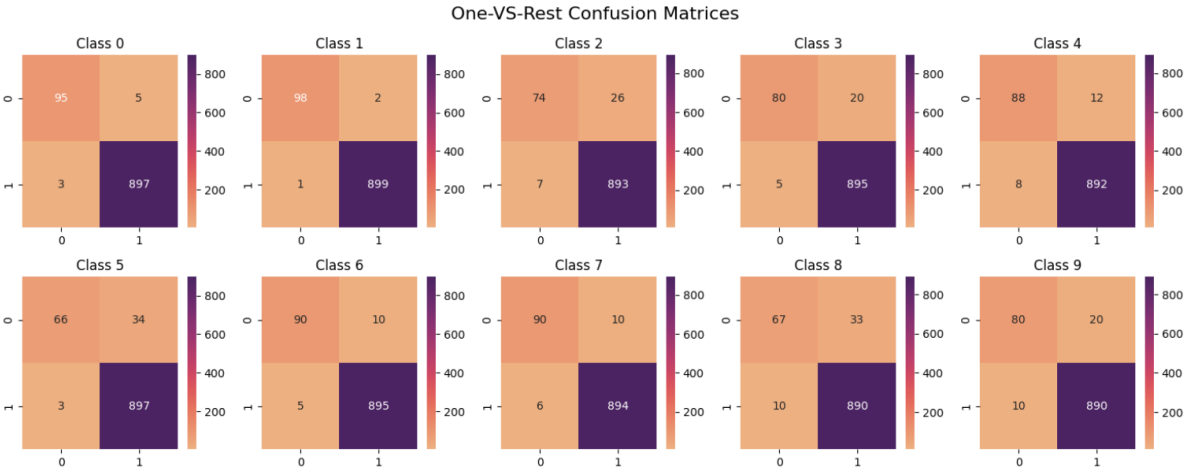
Regarding the hyperplane fitting, by using the one vs rest method, the hyperplane essentially acts as a decision boundary, but it needs to be more deliberate and that is achieved throughout the experimentation and manipulation of certain parameters such as the termination criteria featuring iterations and epsilon., and the cost parameter. By tuning these, the algorithm has more room for better segregation of classes and the hyperplanes are better adjusted. The graph below shows the class-wise accuracies peak and drops.



It seems that so far class 1 had the best recognition so far with an accuracy of 99.7% compared to the rest which is due to the exceptional straight features that the class 1 has against other classes such as complicated digits such as 3 and 4 which are quite distinct from class 1. This class-wise accuracy provides an in-depth representation of how each class performs against the rest of the classes. For example, class 8 had 95.7% accuracy which is high but the lowest against other classes and seems to be getting confused with the other classes considering that the testing data of digits 5,6 and 9 are similar in terms of circular features and 3 horizontal visual features of those digits.

6.4 Confusion Matrix

Below is a one-vs-Rest Confusion Matrices that shows how well SVM is classifying for every class. Class 1 seems to be performing the best with 899 True negatives and 98 True positives, whereas class 8 seems to have a lesser performance level with 43 false values consisting of 33 false positives and 10 false negatives. Nonetheless, the classes got 890 true negatives shows that this leaves spaces for other digits that could potentially be classified as true positives and making the model more accurate in terms of image classification. So far, 7 classes have more than or equal to 80 true positives which is good amount of precision and exactitude for the model.



6.5 Model comparison with K-NN technique

As shown in the image below, the accuracy for KNN and the SVM demonstrates spectacular performances for classifying the digit images. However, if we employ a stringent criterion with an exacting standard, we can see through the inaccuracies and determine the optimal model. For e.g., the performance metrics differ with SVM having several error rates higher than K-NN.

KNN generally shows a higher accuracy overall with 98.1% for class9 and, SVM's One-vs-Rest accuracy drops to 95.7% when it comes to classifying class 8. It should also be noted that when k is 1, the algorithm has a standard accuracy of 94% but SVM's average class-wise accuracy is 97.7%. Even though these numbers aren't quite the same, they do provide a generalisation of how accurate they would be most of the time. K-NN has better precision accuracy and recall compared to SVM's. The lowest bound for K-NN's recall is 0.886 whereas SVM has 0.87 which are both very close. K-NN has better recall and as it can be seen in the image below, SVM classes with higher false positives (FP) such as class 5 has 34 FP resulting with a recall of 0.66 whereas K-NN has 0.92. The metrics also shows that the F1 score for KNN are mostly higher than SVM, where SVM has some values under 0.9 and all K-NN's values are over 0.9 suggesting better harmonic means between precision and recall in K-NN's model. Furthermore, K-NN has better error rates across all classes, and it has 1.9% for class 9 as the highest error rate compared to SVM's error rates of 3.7% and 4.3% for class 5 and 8 respectively. K-NN has also has an error rate of 1.5% and 1.7% for class 5 and 8. This demonstrates that K-NN can determine more complex digits such as 5 and 8, 2.2% and 2.6% more for the respective classes than SVM can.

SVM METRICS

CLASS	TP	FN	FP	TN	One_vs_Rest_accuracy	PRECISION	RECALL	F1 SCORE	ERROR_RATE
0	95	5	3	897	99.20%	0.969	0.95	0.959	0.80%
1	98	2	1	899	99.70%	0.99	0.98	0.985	0.30%
2	74	26	7	893	96.70%	0.914	0.74	0.818	3.30%
3	80	20	5	895	97.50%	0.941	0.8	0.865	2.50%
4	88	12	8	892	98.00%	0.917	0.88	0.898	2.00%
5	66	34	3	897	96.30%	0.957	0.66	0.781	3.70%
6	90	10	5	895	98.50%	0.947	0.9	0.923	1.50%
7	90	10	6	894	98.40%	0.938	0.9	0.919	1.60%
8	67	33	10	890	95.70%	0.87	0.67	0.757	4.30%
9	80	20	10	890	97.00%	0.889	0.8	0.842	3.00%

KNN PERFORMANCE METRICS

Class	TP	FN	FP	TN	STANDARD ACCURACY	PRECISION	RECALL	F1 SCORE	ERROR RATE
0	100	0	5	895	99.5%	0.952	1.0	0.975	0.5%
1	97	3	4	896	99.3%	0.96	0.97	0.965	0.7%
2	87	13	2	898	98.5%	0.978	0.87	0.921	1.5%
3	89	11	7	893	98.2%	0.927	0.89	0.908	1.8%
4	96	4	8	892	98.8%	0.923	0.96	0.941	1.2%
5	92	8	7	893	98.5%	0.929	0.92	0.924	1.5%
6	100	0	5	895	99.5%	0.952	1.0	0.975	0.5%
7	97	3	4	896	99.3%	0.96	0.97	0.965	0.7%
8	89	11	6	894	98.3%	0.937	0.89	0.913	1.7%
9	93	7	12	888	98.1%	0.886	0.93	0.907	1.9%

While not explicitly shown on the table, SVM has more false negatives than K-NN false negatives where K-NN has altogether a total of 60 false negative whereas SVM has 172 false negatives which is almost 3 times larger than K-NN. The fact that it has 116 more false negatives clearly shows the number of misclassifications within the support vector machine model.

Moreover, by the analytical and comparison of SVM and K-NN, it's demonstrated that K-NN offers better performance in terms of class-wise accuracy, precision, recall and F1 score. Due to the endless probabilities of the SVM model through the fine-tuning of hyper-parameters such as the cost parameter and termination criteria (iterations and epsilon) and the fact that it is computationally intensive due to finding the optimal hyperplane that best separates the classes in high-dimensional space shows that K-NN is the better model suited for the digit dataset. Nevertheless, it should be noted that the model adapting to the dataset with this high-level performance can also be signs of overfitting, but as long k has the value of 1 and produces a reasonable accuracy of 94%, it is outstanding in its efficiency.

7 Task 4: Bag of visual words

7.1 Perception Pipeline

The pipeline above shows the computer vision process and steps required for bag of visual words.

Initially, the images are organised in a dictionary where keys are class labels that had been extracted from their filenames and within the data structure contains lists of images that correspond to the labels. This data structure is used to work with the rest of BoVW because it is more efficient and has the labels assigned to them already within the data structure.

The process of Bag of Visual words works by transforming the feature space in numerical form and utilise k-means clustering. The centre points retrieved are the visual words. Then histograms are created for both train and test dataset while extracting local features and comparing them to visual words. 1-NN is thus used to determine the class of the test image by comparing each histogram of train images.

7.2 Feature extraction

Scale-space invariant feature transform (Sift) has been used for bag of visual words and by having a descriptor list, the descriptors can be clustered into visual words. The class-wise separation of descriptors in "sift vectors" helps a lot of the clarification steps that essentially allows efficient mapping of local features to the corresponding visual words in the vocabulary dictionary. The sift vector was also utilised to store descriptors class by class.

Essentially dictionary is essentially used because it offers an O(1) access time which is desirable for time complexity. This is a crucial step as it avoids sorting or possibly filtering descriptors later. In the implementation of Sift below it can be seen that "detectAndCompute" has been used because it is more efficient rather separating the calls for computation and detection. At the very end, the descriptor list and sift vectors are returned for the next part of the algorithm. Another thing to consider for this classification is that not all classes are represented equally because sift examines the pixels nearby the extrema and reject points that have low contrast or poorly defined edges. So, some may have more data points clustered closely while some could be sparse. A single cluster per class would not be able to capture the variations well considering there may be intra-class variability where the variants of certain classes have certain nuances that are very specific to their own class. Inter-class similarity is also the case where certain classes have certain similarities with other classes and those specific nuanced variants could potentially be conflated with one another.

```
def sift_features(images):
    sift_vectors = {}
    descriptor_list = []
    sift = cv.xfeatures2d.SIFT_create()
    for key,value in images.items():
        features = []
        for img in value:
            # keypoint and descriptor
            kp, des = sift.detectAndCompute(img,None)
            if des is not None:
                descriptor_list.extend(des)
                features.append(des)
        sift_vectors[key] = features
    return [descriptor_list, sift_vectors]

sifts = sift_features(train_images)
# Takes the descriptor list which is unordered one
```

This is essentially done for quick facilitation of histogram generation, and they would be used as feature vectors in the BoVW which will be used with 1-NN to make predictions and essentially enhances the model's performance and efficiency.

7.3 Fine-Tuning K-means parameter

```
Average accuracy: %25.251396648044693

Class based accuracies:

0 : %77.6595744680851
1 : %88.67924528301887
2 : %31.25
3 : %18.947368421052634
4 : %21.875
5 : %11.235955056179774
6 : %8.51063829787234
7 : %7.228915662650602
8 : %8.24742268041237
9 : %5.1020408163265305
```

Originally, there were 10 clusters due to having 10 classes for image classification, however in the end it provided with a low accuracy of 25.25%. Considering that we are no longer specifically working with classes but rather descriptors, the initial placement of centroids is usually very sensitive and that leads to ineffective results if there is only one cluster per class. One of the reasons for this accuracy is also due to sparse populated areas where one cluster could impact the model's performance negatively. Because we also have rich descriptors that Sift provides, they

could have been simplified by grouping them into a single centroid per class leading to this low accuracy. Additionally, a cluster centroid might be one step closer to being incorrect to features of another class.

As shown in the code below, the hyper-parameter K was then readjusted to 1000 which offers multiple advantages that has significantly impacted the model's adaptability and performance positively. A high number of clusters have been used to record detailed features that the digits have. 1000 clusters essentially make the model more robust and stronger by containing these features and essentially generalising to new unseen data. This also reduced overfitting considering the dataset of 5000 digits. Moreover, when it comes to advanced sematic segmentation with nuanced features, many clusters such as 1000 can provide a better classification for the class. Increased clustering also outcomes in less quantization errors which mitigates the discrepancy between the original feature vectors and their mapped centroids. Nonetheless, having many clusters have increased overhead and time complexity included in favour of model precision.

```
def kmeans(k, descriptor_list):
    kmeans = KMeans(n_clusters = k, n_init=10)
    kmeans.fit(descriptor_list)
    visual_words = kmeans.cluster_centers_
    return visual_words

# Takes the central points which is visual words
visual_words = kmeans(1000, descriptor_list)
# print(visual_words)
```

✓ 42.6s

7.4 Histogram and 1-NN training

The use of histograms of BoVW acts as feature extraction tool that converts the raw image into a structure quantized form. It allows easier comparison of images for later classifications. Formed by descriptors, the histograms are basically a version of the original image that is composed of the features and significant characteristics of the images with reduced dimensionality.

Another technique that is being used for BoVW is 1-NN which provides another layer of performance. It works very well, which classifies given a data point on its closest neighbours from its training data with a feature space that is well-defined and controlled. 1-NN classifies images from the testing data based off the previous trained visual words obtained from the centres of the centroid. It matches each test histogram data with the training image histogram and assigns a probabilistic label to the nearest neighbour.

In terms of computational performance and complexity, a histogram significantly decreases the amount of data that needs to be processed, however, in the trade off from time complexity as the number of clusters increased and usually takes longer because there's more features being detected and stored with the number of clusters. Nonetheless, the quantization of data into histogram makes this process feasible upholding a higher accuracy while maintaining lower memory usage.

In summary, the BoVW histograms effectively contains the image content capturing the necessary features while 1-NN holds the classification test which results in a more efficient and optimised pipeline for image classification.

7.5 Accuracy and performance metrics

```
Average accuracy: %57.31843575418994

Class based accuracies:

0 : %84.04255319148936
1 : %90.56603773584906
2 : %42.70833333333333
3 : %64.21052631578948
4 : %77.08333333333334
5 : %50.56179775280899
6 : %50.0
7 : %32.53012048192771
8 : %50.51546391752577
9 : %42.857142857142854
```

According to the accuracies performed from every class-based item, the model shows a notable variance ranging from as low as 32.5% for class 7 and as high as 90.5% for class 1. Even though the model yields an average accuracy of 57% demonstrating predictive power, the model has room for great improvement. Considering number 0 and 1 have simple visual features, it could be easy for the model to classify them as compared to difficult classes such as 5 and 7. Class 6 and class 9 could also possibly be conflated with each other showing an accuracy of 50% and 42.8% respectively. Perhaps the inconsistency could be attributed to hyper parameter such as k that also initialises the centroids could be learning too many features and essentially be learning the noise which is a sign of overfitting. Which concludes why a lot of classes have low accuracies such as class 2,5,6,7,8 and 9. Moreover because Sift takes feature descriptors from every image, the number of white pixels depending on how different and nuanced the variant digit could also matter because only credible extrema that have good edges and features are selected. So, this inconsistency of different number of features per class based on the gradient magnitude $\sqrt{{G_x}^2 + {G_y}^2}$ where the strength of the vector at every pixel where the digits have their visual footprint against the black background shows that there's an ambiguous number of key points being selected per image and per class which leads to imbalanced data. Although if a lot of clusters out of a 1000, contains features from different classes, essentially indicates that the clustering method or cluster does not have enough predictive and discriminative power to segregate different variants of testing classes. In summary, this shows how this complex procedure with multiple algorithms and certain hyper-parameters impacts the accuracy of a multi-class classification.

Perhaps an improvement for next time would be denser SIFT features which will improve accuracy because it can capture more granular details within the pixelated image. This would allow the models to have more options and versatility for class differentiation due to the richer more discriminative visual vocabulary. Although, there is an increase in computational cost and the data would still be imbalanced due to SIFT extrema selection criteria.

8 Task 5: Convolutional neural networks

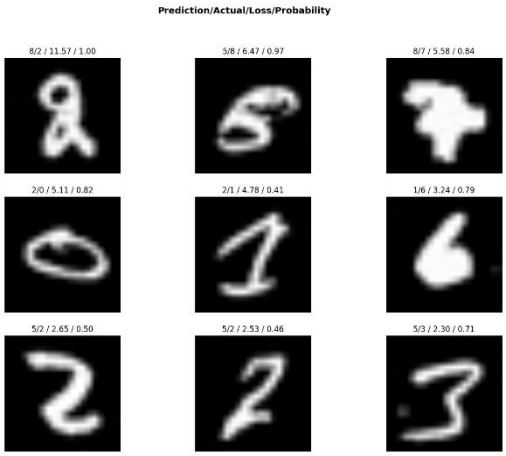
8.1 Alex net

8.1.1 Training and regularisation (Data Augmentation), and Validation Loss

epoch	train_loss	valid_loss	error_rate	time
0	1.833637	0.365744	0.118750	00:23
1	0.847370	0.201353	0.060000	00:20
2	0.496316	0.176595	0.050000	00:17
3	0.354670	0.158350	0.043750	00:22

This time utilising convolution neural networks such as Alex net to train on the dataset shows improvement on the train loss and validation loss which is an indication of the model learning and adapting to the dataset very effectively.

However, there's still improvement to be made as seen in the error rate of 0.04 in the last epoch. In the most confused chart, shows specific digit pairs such as '5' and '3' that are typically misclassified. For example, a poorly written '5' can resemble a '3' and the model would need more time to record and train upon this type of data to differentiate them. This differentiation was done by regularisation so that it trains on the features that it's missing.



By incorporating regularisation through Data Augmentation below, there is a significant increase in both train loss and valid loss also with a higher error rate. Even though augmentation was supposed to make generalisation better, the model might have become too complex, and the augmentations were too much for the model to take on which may have introduced variability to the training data. But the validation loss as a hyper-parameter selection has shown consistent decline from 0.574 to 0.2954 over 4 epochs. Nonetheless, the error rate has increased from 0.08 to 0.09 in the last two epochs suggesting that the model will need more fine-tuning to fully adapt to the nuances that certain digits have.

epoch	train_loss	valid_loss	error_rate	time
0	2.175023	0.574271	0.171250	00:27
1	1.136379	0.352516	0.103750	00:34
2	0.754286	0.302408	0.085000	00:36
3	0.640896	0.294928	0.091250	00:25

8.1.2 Accuracy and Testing Data

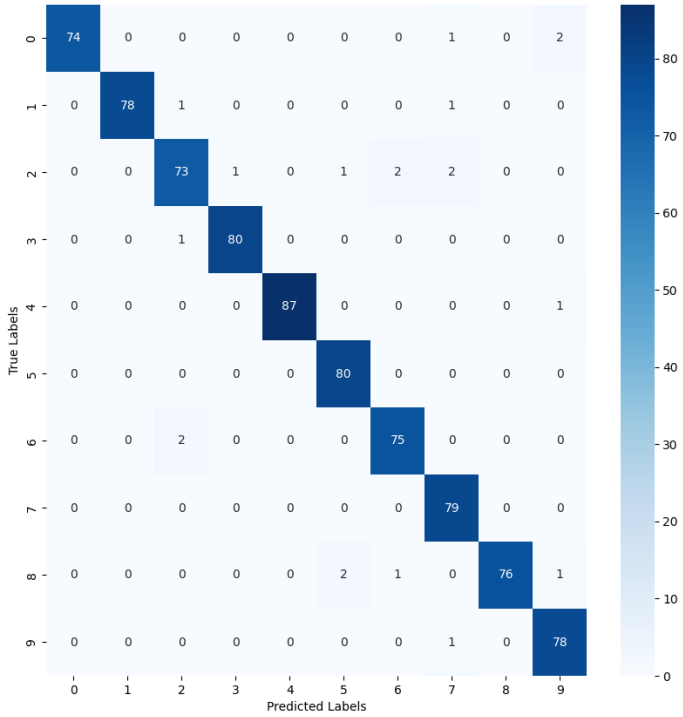
	Class	Accuracy
0	0	0.975000
1	1	0.975610
2	2	0.906977
3	3	0.973684
4	4	0.986842
5	5	0.932584
6	6	0.987805
7	7	0.971831
8	8	0.951220
9	9	0.973684

Precision: 0.9630988788285931
Recall: 0.9635236672421712
F1 Score: 0.9631318846732013

In terms of accuracy, the training metrics indicate a high-performing model with an average of 96%. The precision and recall also have about 96.3% accuracy which shows that the model determines the true positives well including false negatives and false positives as well. But for the class-wise accuracy, class 2 and 5 have the lowest accuracy which aligns with the most confused chart. These digits were often confused however, a minimum of 90% is still relatively good even though it still needs some fine-tuning.

Upon experimenting on the testing data, the accuracies are reasonably well and are about 97% (1% better) than testing on the validation set. The high recall and precision on the testing data are very high indicating that the model is capturing the true positives via recall and not misclassifying other digits as the true class through precision.

```
Precision:  0.9752795854936123
Recall:  0.9747748024752244
F1 Score:  0.9747276948274329
```



Moreover, the confusion matrix shows the detail and predictions the model made on each class and there are very few mistakes that the model is really making such as mistaking class 1 for a '7' and sometimes if class 7 is written very poorly or very thin, technically it could be considered as a 1 even by most people. So little mistakes where certain classes are written in a way that genuinely look like other classes are acceptable for this model given the fact that most of the time, the class would be written capturing most of the primary features of the digit itself.

Furthermore, through the experimentation of the model including hyper-parameter selection and fine-tuning epoch as well as using data augmentation to have a reflection on the dataset and the model's learning behaviour, it can be seen that Alex Net produces high performance when tested on the testing data Therefore Alex Net can be deemed an appropriate pre-trained convolution neural network that can be used for image classification.

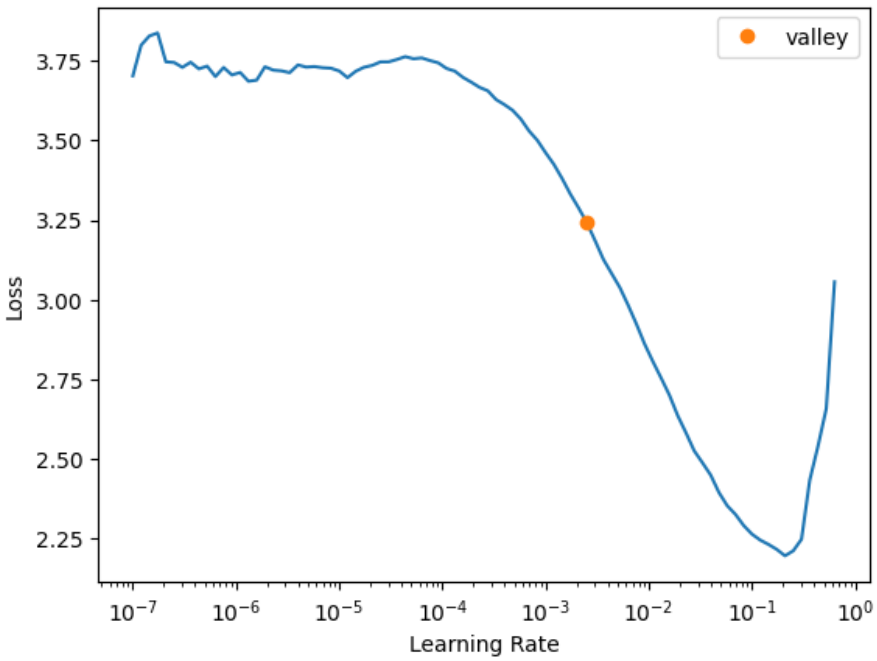
8.2 VGG16

8.2.1 Training and regularisation (Data Augmentation)

epoch	train_loss	valid_loss	error_rate	time
0	1.790454	0.357230	0.117500	00:50
1	0.739675	0.167301	0.050000	00:42
2	0.365967	0.133294	0.045000	00:42
3	0.215729	0.124569	0.040000	00:41

VGG16 is another pre-trained convolution neural networks that can be used for the digits dataset. Initally it had a final train loss of 0.2 over 4 epoch, perfomed 0.1 on the validation set resulting an error rate of 0.04 which is good but can still be improved. By regularising the data, perhaps a better error rate can be expected however, in the first epoch itself it had a train loss of 2.17 which is 0.8 more than the initial train loss when the dataset was initally trained on. In the last epoch, the error rate has gotten worse which shows that this model cannot adpat to new augmented data as it has become too complex at this point and learning scaled features of the nuanced digits can be very challenging. To be able to combat this low accuracy and to improve generalisation, a learning schedular was applied to find the learning rate. By fine-tuning this model, it may find diverge a little bit because at this point a small learning rate at the very start of the model is not performing of the standard that's required. So the learning rate graph below shows where the loss is at descending.

epoch	train_loss	valid_loss	error_rate	time
0	2.090266	0.587880	0.206250	00:47
1	1.089607	0.257909	0.080000	00:47
2	0.684430	0.207375	0.070000	00:49
3	0.498919	0.203668	0.068750	00:47

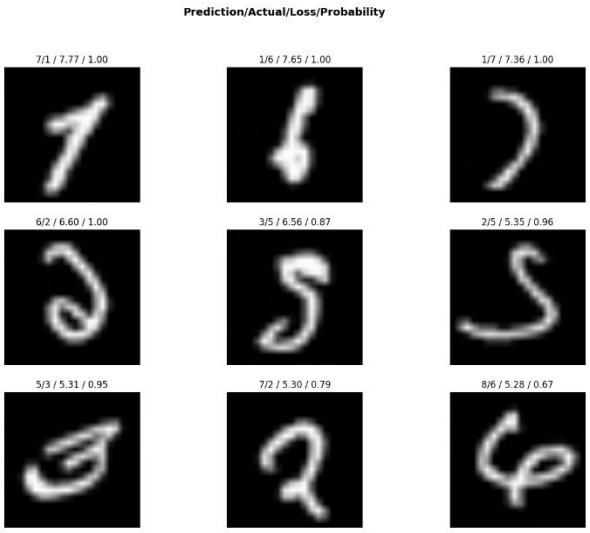


8.2.2 Parameter Tuning, Training Scheduler and Fine-tuning

epoch	train_loss	valid_loss	error_rate	time
0	1.730549	371.596100	0.811250	01:00
1	1.395498	0.504255	0.102500	00:59
2	0.790555	0.341096	0.075000	01:00
3	0.475989	0.179046	0.051250	01:01

Applying a suitable learning rate is important so that it does not converge and start overfitting noise and never ends. Applying between 10^{-2} and 10^{-1} was convenient because that's where it decreases the most. 10^{-0} was avoided because that's where the slope picked back up and increases significantly. But experimenting with the learning schedular, the learning rate was finalised and fine-tuned due to the significant decrease in most importantly the validation loss and error rate demonstrating the model's learning rate shaping its prediction .

8.2.3 Validation data and Misclassifications



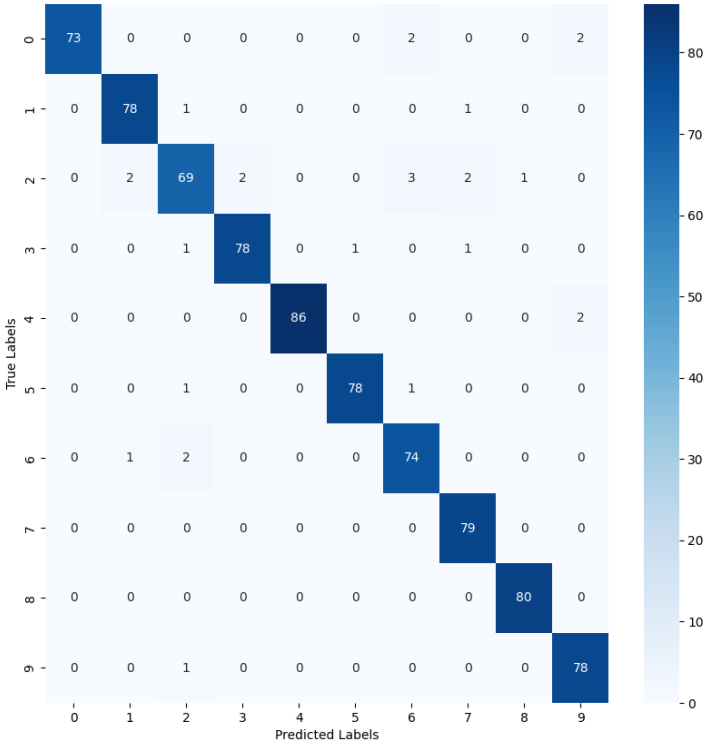
While testing on the validation data, it has got Class 9 100% correct which is an indicative standard of the model. Class 2 had the lowest accuracy of 85% indicating the model is weak on class 2 variants. According to the false predictions, it shows that class 2 was confused with class 6 of which the written '2' would have technically looked like class 6 if it was flipped on a horizontal scale demonstrating the model recognising the features, but just for an incorrect class.

8.2.4 Testing data Accuracy and confusion matrix

	Class	Accuracy
0	0	0.948052
1	1	0.975000
2	2	0.873418
3	3	0.962963
4	4	0.977273
5	5	0.975000
6	6	0.961039
7	7	1.000000
8	8	1.000000
9	9	0.987342

Precision: 0.9660985797213302
Recall: 0.9660086092997485
F1 Score: 0.9657679971258764

Performing on the testing data shows great performance such as having 0.96 accuracy for precision demonstrating that Its does not misclassify other digits that much. It also has an F1 score of 0.965 which has a good distribution between precision and recall showing that the model can capture and classify true positives and also not misclassify other digits incorrectly. The confusion matrix below shows the class-wise accuracy. The model classified all of class 7 and class 8 with 100% accuracy of which Alex net was only able to perform 100% on one class.



As the model keeps getting fine-tuned and adapts to a learning rate that diverges a bit at the start, it will keep resulting in high-performance for digit recognition.

8.3 Residual Networks

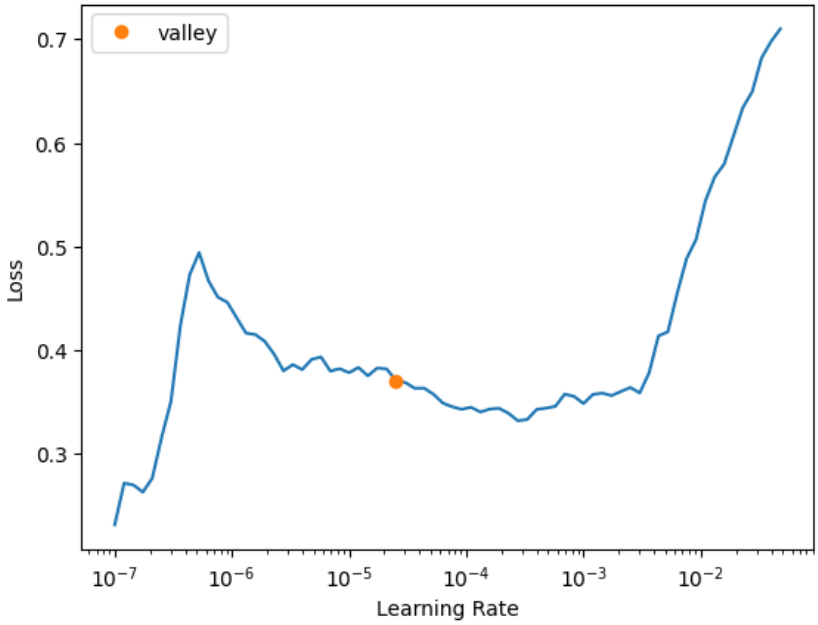
8.3.1 Training, Learning Rate and Fine-tuning

Residual Networks is another deep learning architecture that trains extremely deep networks and has a lot more layers than Alex-net and VGG16.

epoch	train_loss	valid_loss	error_rate	time
0	2.145350	0.903836	0.327500	00:43
1	1.011290	0.304331	0.097500	00:40
2	0.613050	0.218945	0.067500	00:40
3	0.454349	0.208515	0.063750	00:40

Upon the initial training of ResNet over 4 epochs, it resulted in a train loss of 0.4 and a validation loss of 0.2 resulting an error rate of 0.06. The model needs to be trained more, epoch wise so that it keeps training the last layer and a lot of fine-tuning will need to be done so the model can yield high performance. Furthermore, the “unfreeze()” function was used to unfreeze the earlier layers and allow backpropagation to adjust the weights which will make the network learn from its errors which is better than regularisation. The first epoch had an error rate of 0.07 as compared to the first epoch of the initial training which is 0.3 demonstrating this significant difference in false predictions. Since, the model is learning from its errors, it is contingent that an appropriate learning rate is set so the model does not completely diverge as this large learning rate results in errors.

epoch	train_loss	valid_loss	error_rate	time
0	0.491019	0.233121	0.076250	00:44



8.3.2 Parameter Tuning and Training Scheduler

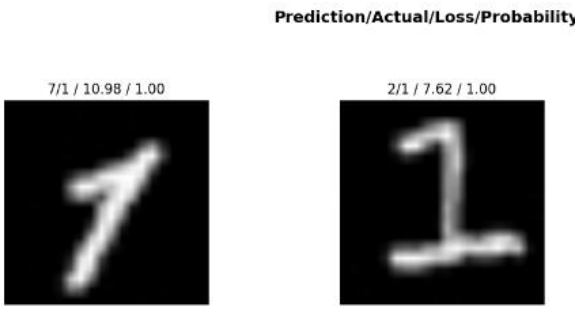
20	0.020573	0.058217	0.013750	00:31
21	0.016058	0.059491	0.016250	00:31
22	0.020191	0.058930	0.013750	00:31
23	0.019426	0.059074	0.017500	00:30
24	0.017814	0.057208	0.012500	00:31

By choosing a learning rate between 10^{-5} and 10^{-4} and training the model over 25 epochs, the model can keep learning new features and by testing on the validation data, it won't be a biased model that only trains and test from a singular perspective. On the last epoch, it has a train loss of 0.017, a validation loss of 0.05 with an error rate of 0.01 of which is very low indicating great performance upon the training phase.

8.3.3 Accuracy and confused classes

```
   Class  Accuracy
0      0  1.000000
1      1  0.962963
2      2  0.985714
3      3  1.000000
4      4  0.988506
5      5  0.978261
6      6  0.988506
7      7  1.000000
8      8  0.987013
9      9  0.985507
Precision:  0.9872159372159371
Recall:  0.987646984588514
F1 Score:  0.9872742939127834
```

Reviewing the performance metrics, it can be seen there is an accuracy of at least more than 96%, and mostly 98% in the class-wise accuracy. The residual model has successfully classified class 0, class 3, and class 7 with 100% accuracy on the validation data displaying optimal performance so far. Both precision and recall have 0.987 accuracy connoting its ability to detect true positives is performing at a very high level and rarely misclassifies other digits which is tremendously of a very high standard for the validation set itself. However, class 1 had the lowest accuracy pinpointing at 0.96, which is very good that it's above 95%. Nonetheless, investigating its inaccuracy below shows that it gets confused with a variant of class 7 and class 2. If both of these poorly written classes were blurred and perceived from a distance to the human eye, it would seem to be true as



there are certain features that's mainly visually composed of the other classes even though it's not. Nevertheless, it's inclined to overlook this trivial error.

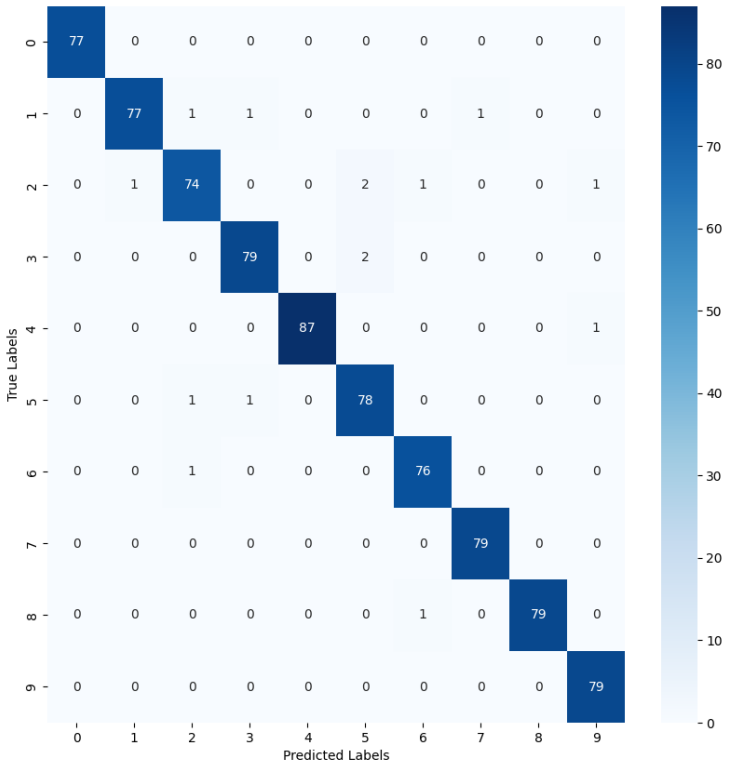
8.3.4 Testing data Accuracy and confusion matrix

	Class	Accuracy
0	0	1.000000
1	1	0.962500
2	2	0.936709
3	3	0.975309
4	4	0.988636
5	5	0.975000
6	6	0.987013
7	7	1.000000
8	8	0.987500
9	9	1.000000

Precision: 0.9811914218723162
Recall: 0.9812666853384153
F1 Score: 0.9811525357183853

Upon testing on the testing dataset, class 1 and class 7 still had 100% accuracy, however class 3's accuracy was reduced by 2.5% and class 9 has a 2.5% in accuracy resulting in a 100% accuracy. This shows that class specific features have been learned exceptionally well. The overall Precision, recall and F1 scores are around 98.1% accuracy demonstrating high predictive power and the model's reliability. This demonstrates that the model is assertive in its classification and is neither a conservative model nor a liberal model which is good because it won't result in higher false positives nor lower false negatives respectively.

Due to ResNet's ability to generalise well due to their deep learning architecture and structured residual connnections, the performance metrics affirm this. Class 2 have the lowest accuracy of 93.6% could have been due to overfitting and perhaps some of the unessary features of class 2 are being learnt and that noise has been propagating. According to the confusion matrix, the model has been confusing 5's with 2's. From an artistic perspective, if class 5 was to be written similar to an 's', the horizontally flipped version would contain close features as class 2 and almost identical which explains that the model is often making confuion between some indepth features of class 5 and class 2 wrong which results in an incorrect classsification of class 2 variants.



8.4 CNN Comparison

AlexNet, VGG16 and ResNet are all pre-trained CNN that are designed for image classification, and they all differ in terms of architecture, number of layers, depth, performance, and behaviour. AlexNet has 5 convolution layers followed by 3 fully connect layers and uses a rectified linear unit (ReLU), an activation function that is trained to make the network learn from its error by making all the negative pixels to 0. Whereas VGG16 is deeper than AlexNet with 16 layers but it uses a smaller (3x3) filter for multiple layers. ResNet are also very deep with layers ranging from 18 to 152 layers.

Model	Performance Metrics	Accuracy
AlexNet	Precision	0.9752795854936123
	Recall	0.9747748024752244
	F1 Score	0.9747276948274329
VGG16	Precision	0.9660985797213302
	Recall	0.9660086092997485
	F1 Score	0.9657679971258764
ResNet	Precision	0.9811914218723162
	Recall	0.9812666853384153
	F1 Score	0.9811525357183853

AlexNet has an F1 score of 97.4% showing a good balance between precision and recall and classified class 5 and 7 with 100% accuracy which demonstrates its reliability towards certain digits. Whereas VGG16 has an F1 score of 96.5% which is almost 1% less than AlexNet and classified class 7 and 8 perfectly. In terms of comparison, it seems that both models are performing reasonably well for the dataset but technically AlexNet seems to be resonating, beating VGG16 with about 1%, performance wise.

ResNet overall had the best performance across the classification evaluations resulting in an accuracy of 98.1% which is better than AlexNet and VGG16 due to the large number if layers that it has been trained on without the issue of the vanishing gradient due to the skip connection and residual connections. It also classified three classes perfectly of which AlexNet and VGG16 could only classify two classes with zero error. ResNet classified class 0,7 and 9 with 100% accuracy. It appears that out of all CNN models ResNet adapts the most to the dataset and can capture more complex features due to its architecture and fine-tuning phase, therefore achieving higher accuracy for most digits.

Hence, Residual Network is the most effective CNN model for digit recognition as it can capture all the nuanced features of the written digits and its deep layers aid a lot in classifications. Furthermore, ResNets rarely ever gets perplexed and will still classify the image correctly by recalling the primary visual characteristics and features of each class.

9 Bibliography

FastAI. (2023). *Welcome to fastai*. Retrieved from Fast AI Documentation: https://docs.fast.ai/

mailchimp. (2023). *PNG vs JPG: What You Should Know*. Retrieved from mailchimp: https://mailchimp.com/resources/png-vs-jpg/#:~:text=If%20you%20want%20to%20have,the%20compression%20image%20is%20done.

Yalçıner, A. (2019, July 13). *Bag of Visual Words(BoVW)*. Retrieved from Medium: https://medium.com/@aybukeyalcinerr/bag-of-visual-words-bovw-db9500331b2f

10 Appendices