University of Cape Town

EEE4114F

Digital Signal Processing

Hand-drawn Shape Classification

Author:
Murray Inglis

Student Number:
INGMUR002

July 25, 2024

Plagiarism Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

July 25, 2024

Murray Inglis

Date

Contents

1 Problem Statement

The shape classification problem is where a discrete class of shape is assigned to an input image. The input image must be assigned one label from a set of labels that best fits the image. This is a rather trivial task in terms of human intelligence, however the task becomes more complex when implementing it in terms of computer vision due to how the image is represented and viewed by a computer.

2 Introduction

This report details the design and implementation of a Convolution Neural Network to classify hand-drawn shapes. The model will be trained with the aim to be able to classify 5 different shapes. Namely, these shapes are: circle, triangle, square, pentagon, and hexagon.

3 Literature Review

3.1 Model-Based vs Data-Based Approaches

Two approaches for classifying shapes can be discerned, a model-based approach and a data-based approach. The model-based approach would involve finding or defining your own algorithm that mathematically maps an input to an output. A data-based approach would involve accumulating a dataset of images that can be used for training. The model will be able to classify the input based on the dataset that it trained on [1]. Each approach has its caveats and advantages.

The model-based approach clearly involves having to develop an algorithm for your problem. This algorithm may prove extremely difficult and complex to develop. This approach requires knowledge within the domain of the problem. You will have to leverage your understanding of the problem to develop some sort of relationship or set of equations. With hand-drawn shape classification, an algorithm could be developed to locate the vertices of the shape on the image. Other factors, such as the hand-drawn lines overlapping at the points where they touch. As detailed in an article by *Zhang* [9], a pre-processing filter has to be applied to smooth out all the edges. On the other hand, if the algorithm works as specified, then the model may be much more computationally efficient than a data-based approach.

The data-based approach has the problem of relying heavily on the collected dataset used for training. The process of data collection is key here and must consider many possibilities for how the model is intended to be applied. The data collected should fully express the features of the data that the model will perform on once training is completed. With the data-based approach, there is the advantage of flexibility and adaptability. In-depth knowledge of the field that the data lies in isn't required, as long as a sufficient amount and representation of data is collected. This however leads to the main disadvantage of a data-based approach; that if there

is insufficient data available, then performance will be severely impacted. Choosing the right approach will take into account all of these factors.

3.2 Model Selection

The most common and effective model used for image classification is the Convolutional Neural Networks (CNN), due to its inherent fit for computer vision. The K-Nearest Neighbours (KNN) will also be considered. A CNN is designed to process data that is in the form of multiple arrays. The CNN takes advantage of properties that are typical of natural signals, as according to an article written by *Bengio* [8, p. 439], these are: "local connections, shared weights, pooling and the use of many layers." Higher level features are obtained through the composition of lower level features. The lower layers of the CNN will extract the low-level features of the input, such as basic lines and vertices. The final layers will perform the more complex task of combining the features to identify the pattern and perform the classification task at hand. This use of layering allows for better generalisation. The CNN isn't reliant on specific pixel patterns, but rather the underlying feature. This allows it to be invariant to shifts and distortions. The convolutional layer detects the features and the pooling layer combines the features.
A KNN classifier is not often used for image classification. For KNN, the entire image dataset is required to be stored during runtime. This is often not computationally viable due to the size of the dataset. KNN finds the K most similar images in the training data to the input image. A distance metric, such as Euclidian distance, is used for this. The image is classified as the label that is the majority of its K nearest neighbours. As the dimensionality of the data increases, the classifier is limited. Each pixel contributes a feature, making an image dataset computationally intense for a KNN classifier. Due to the KNN classifier relying on having access to the dataset at all times, it hasn't learnt the classification task. This means that it won't generalise well. A KNN is not typically suited for image classification as compared to a CNN, primarily due to the high dimensionality of the images. The convolutional and pooling layers of the CNN that allow it to process subsets of the input at a time are more suited for image classification.

3.3 Data Augmentation and Feature Reduction

The concept of data augmentation and regularisation through feature reduction are to prevent the model from underfitting or overfitting, helping the model generalise better. Underfitting is when the model is too simple to understand the relationships between input features. Overfitting is when the model becomes too complex and captures features that are irrelevant.

As discussed in an article by *Antunes et al* [6], data augmentation is a useful way to both increase the quantity of training data and the robustness of the model. Adding noise to the data, as well as transformations such as rotations and shifts, allow a completely new data point to be created. This article also discusses the use of normalising the data with a transformation

of subtracting the mean from the values of a dataset and dividing the values by the standard deviation. This effectively centres the data around 0 and gives it a unit variance. This allows for faster and more stable training, since high value features won't dominate as much.

The use of regularisation is also used to forcibly simplify the model. This allows for a complex model, while preventing overfitting. A common method of regularisation is to penalise weights that become large. As explained in *A Survey of Techniques All Classifiers Can Learn from Deep Networks: Models, Optimizations, and Regularization* by *A. Ghods* and *D. J. Cook* [7], the parameter norm penalty (or p-norm penalty) function have two popular implementations: the L1 and L2 regularisation. The difference between the two being that L1 sets features to 0, whereas L2 just minimises weights towards 0. Dropout is another popular regularisation method. Dropout randomly deactivates a set of input nodes and nodes in the hidden layers. This prevent the model from memorising the training data and therefore prevents overfitting. Optimisation techniques are used to backpropogate the errors in the network to update the weights. This is done using variations of gradient descent. Examples mentioned in the article by *A. Ghods* and *D. J. Cook* include stochastic gradient descent, mini-batch gradient descent, momentum, Ada-Grad, and Adam.

4 Methodology

4.1 Libraries Used

Firstly, the libraries used for making and training the model were PyTorch and sklearn [5, 3]. There are many publicly available resources and tutorials for using this python packages. Additionally, CUDA was used for training the model on my laptop's NVIDIA graphics hard [2].

4.2 Dataset creation

A labelled dataset is created, since a supervised learning method will be used. Each image is named with their label at the start and then and index at the end. For example:

circle1.png
circle2.png
triangle1.png
...

This allows for a function to be created to easily read in the names of every image file and determine the label for that file. A csv is created containing this information. The csv has the format:

circle1.png,0

circle2.png,0
triangle1.png,2
...

Label encoding, or ordinal encoding, is used for encoding the labels of the shapes. The label map is:

circle $\rightarrow$ 0
square $\rightarrow$ 1
triangle $\rightarrow$ 2
pentagon $\rightarrow$ 3
hexagon $\rightarrow$ 4

Consideration is taken into adding variance to the data used to train and test the model. In order to try make the model as accurate possible, a basic python app was designed to take in drawings of shapes. This app has modes that can be set in order to automatically label the data. By using shapes drawn by different people, there is increased variance in the data provided to the model.

An augmentation function is created that applies each of the following transformations to every image in the dataset:

- Rotation by a random angle between -180 and 180 degrees
- Randomly perform a horizontal or vertical flip
- Scaling by a random factor from 0.5 to 2
- Adding Gaussian noise with a random normal from 0.5 to 2

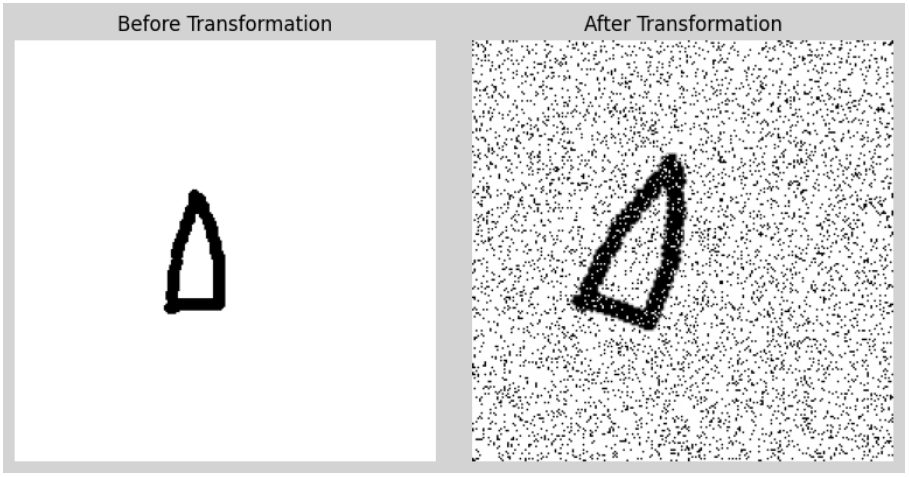

Figure 1: Transformation Comparison

The noise from this augmentation was found to have a significant effect on the model. In the Section 5 section, this effect can be seen better.

The dataset contains 40 of each shape originally. Since 20 transformations are done to each image, this creates a dataset of size 4000. In addition, a separate testing dataset of size 300 is created (also using the transformation function). This dataset is unseen for the training phase and is only used at the very end to finalise the models validation.

A normalisation process is used for the training data before they are converted to tensors. The images are first converted to greyscale to reduce the number of features. The numerical sum of the pixel values of the entire dataset is calculated and used to determine a mean and standard deviation for the pixel values. The images are transformed to tensors. The mean is subtracted from each pixel value and the pixel value is divided by the standard deviation. By normalising the input data to within a similar scale range for each channel, it allows the model to learn more effectively.

Figure 2: Shape Classes

4.3 Model Selection

There is the option to choose a pretrained neural network that can simply be imported from a python package and trained, or there is the option to define your own network. Each approach has its own advantages and disadvantages. A pretrained model already learned useful features from training on a large dataset. By leveraging these pretrained features, they can be transferred to this task. They can also be fine-tuned to meet specifications, however for this investigation they were not. For the purposes of this project, a CNN will be built by defining each layer manually. My own defined CNN will have its performance compared to the pretrained model. The pretrained model used was *resnet18*. This is a CNN that is 18 layers deep. Further information can be found in the PyTorch documentation [4].

A Convolution Neural Network (CNN) is used as the classification model I will be defining. A CNN is particularly suitable for this type of problem. The CNN can process large amounts of data and produce an accurate prediction. An image is an input with a large amount of data,

and considering there are many images as inputs, there is a very large amount of data. The CNN chosen has the following layers:

- 5 convolution layers
- A max pooling layer after each convolutional layer
- 1 dropout layer after the last pooling layer
- A ReLu activation function is applied after each convolutional layer and the first two fully-connected layers
- 3 fully-connected linear layers at the end of the network

Since the input batch of images are grayscale, the first convolution layer receives 1 channel as input. This first layer outputs 32 channels. Shown below is the exact structure of the model as defined in the code:

```
class SimpleCNN(nn.Module):
def __init__(self, num_classes=5):
    super(SimpleCNN, self).__init__()
    self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
    self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
    self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
    self.conv4 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
    self.conv5 = nn.Conv2d(256, 512, kernel_size=3, padding=1)
    self.dropout1 = nn.Dropout(0.5)
    self.fc1 = nn.Linear(512 * 7 * 7, 1024)
    self.fc2 = nn.Linear(1024, 512)
    self.fc3 = nn.Linear(512, num_classes)
    self.relu = nn.ReLU()
    self.pool = nn.MaxPool2d(2, 2)

def forward(self, x):
    x = self.pool(self.relu(self.conv1(x)))
    x = self.pool(self.relu(self.conv2(x)))
    x = self.pool(self.relu(self.conv3(x)))
    x = self.pool(self.relu(self.conv4(x)))
    x = self.pool(self.relu(self.conv5(x)))
    x = torch.flatten(x, 1)
    x = self.dropout1(x)
    x = self.relu(self.fc1(x))
    x = self.relu(self.fc2(x))
    x = self.fc3(x)
```

```
return x
```

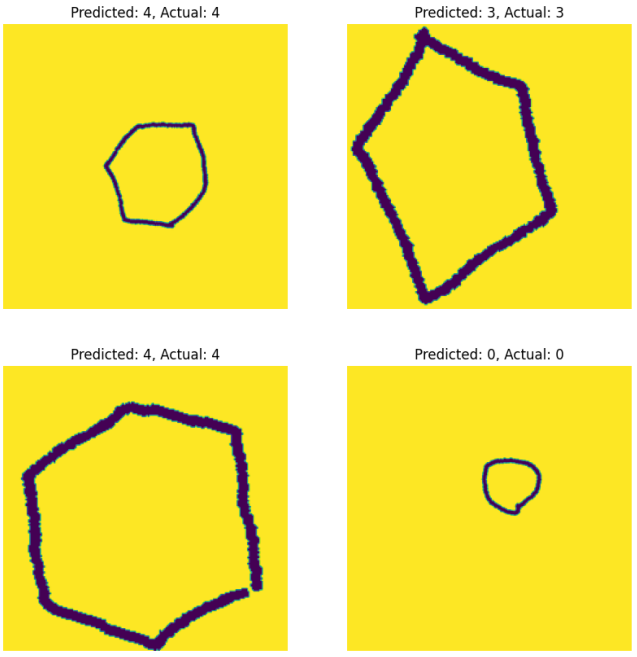

Figure 3: Example of the model's predictions

4.4 Splits Used and Validation

To train and validate the model properly, an informal grid search process was used. This process manually tuned the learning rate and batch size to different values and comparing the performance each time. These were varied according to the following values:

- Learning rate: [0.001, 0.01, 0.1]
- Batch Size: [8, 16, 32]

Through this tuning process, a learning rate of 0.001 and a batch size of 16 were found to have the best results and these were used for the final CNN model. Additionally, K-Fold Cross-Validation was used for the validation process. Folds of 2, 3, and 5 were used. Each fold was trained through 30 epochs. An early stopping condition was added also. If the validation loss had not improved for 10 epochs, then the model training would be stopped early. For each K value, the training accuracy and validation accuracy of each fold are plotted against the epoch number. The training loss and validation loss are also plotted against the epoch number. This helps visualise the training process and helps determine which hyperparameters need to be tweaked. It allows you to see overfitting and underfitting and whether the model was sensitive to a specific split of the data.

4.5 Optimisation and Regularisation

For the gradient descent optimiser, Adam is used. As mentioned in Section 4.4, a learning rate of 0.001 was found to be most optimal for the optimiser. L2 regularisation was used on the features, with an L2 lambda of 0.01.

5 Results and Validation

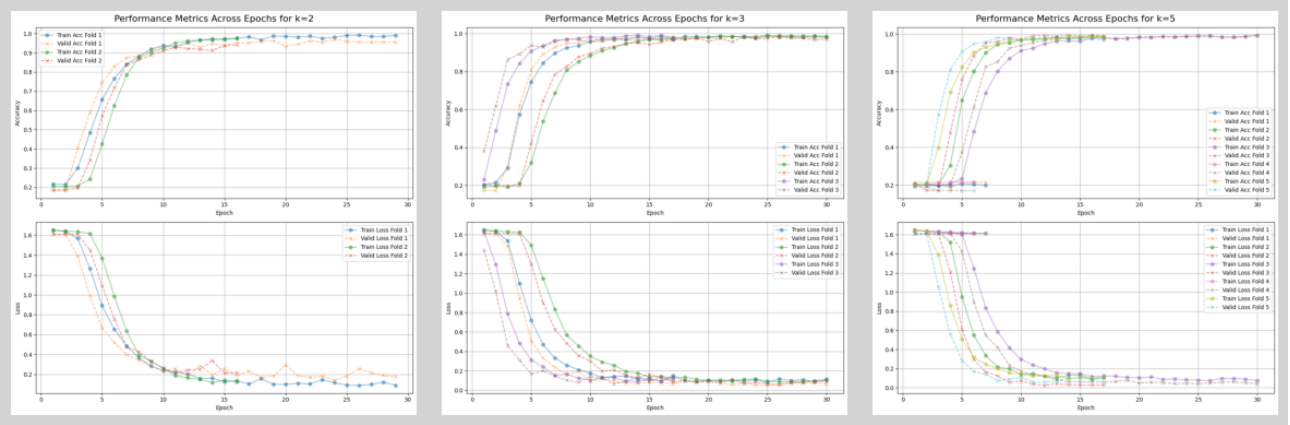

Figure 4: Metrics for model trained on small dataset without augmentation

K	Fold	Accuracy
2	0	0.472
2	1	0.472
3	0	0.540
3	1	0.445
3	2	0.459
5	0	0.189
5	1	0.459
5	2	0.445
5	3	0.189
5	4	0.486

Table 1: Small dataset without augmentation performance on unseen data

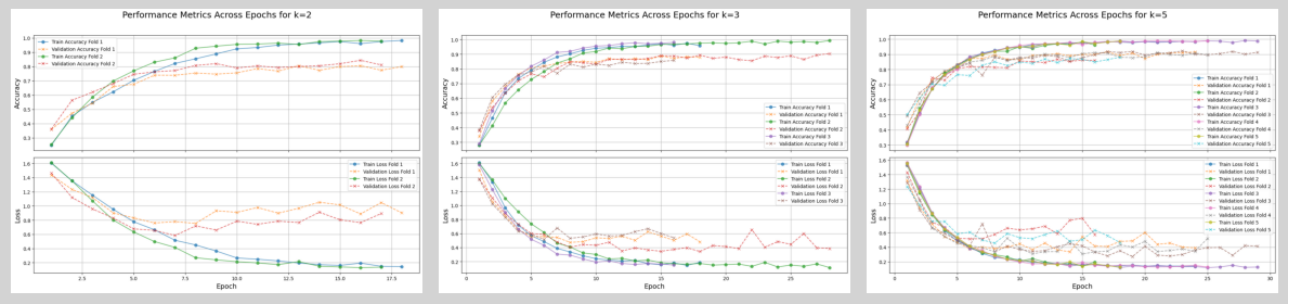

Figure 5: Metrics for model trained on dataset with augmentation (no noise)

K	Fold	Accuracy
2	0	0.729
2	1	0.810
3	0	0.783
3	1	0.851
3	2	0.783
5	0	0.851
5	1	0.810
5	2	0.918
5	3	0.878
5	4	0.770

Table 2: Dataset with augmentation (no noise) performance on unseen data

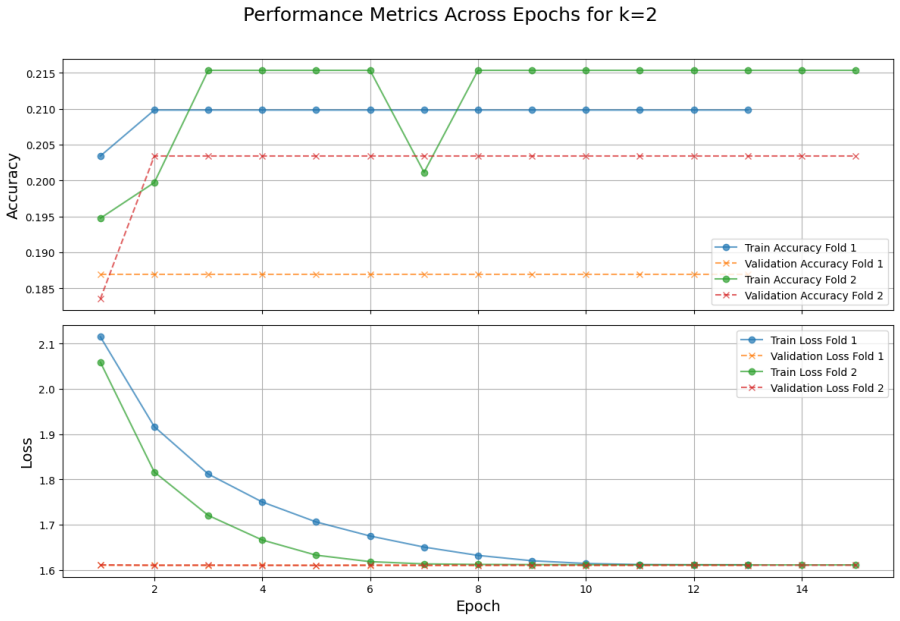

Figure 6: Metrics for dataset with noise

The K-Fold training was ended after k=2 for the dataset with noise due to its poor performance. The three different datasets used to train the same model show vastly different results. This shows the importance of the validation process of machine learning.

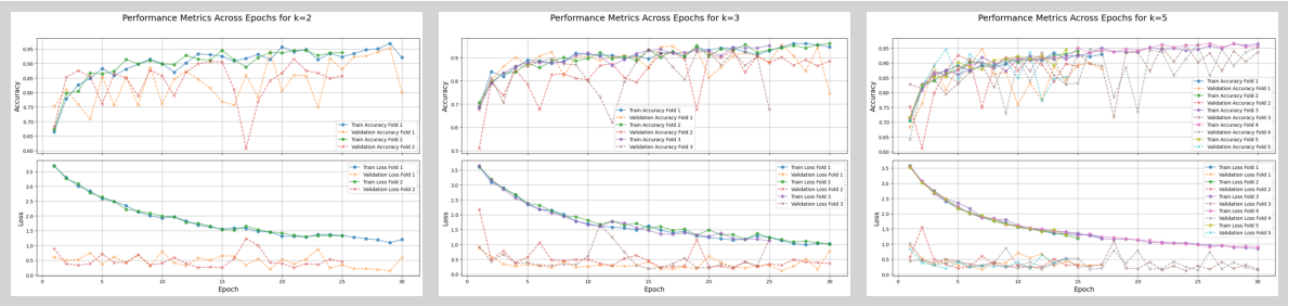

Figure 7: Metrics for resnet18 (trained on noisy data)

K	Fold	Accuracy
2	0	0.756
2	1	0.864
3	0	0.729
3	1	0.837
3	2	0.621
5	0	0.824
5	1	0.905
5	2	0.918
5	3	0.905
5	4	0.783

Table 3: resnet18 performance on unseen data

It can clearly be seen that resnet18 has outperformed the other models. My CNN cannot handle noise, having no increase in improvement. It does not even manage to train on this data. It has about the same accuracy as randomly choosing 1 out of 5 of the shapes after training. resnet18 performs well on noisy data. It has generalised well and has high accuracy for the tests on the unseen data. It does exhibit more variance than the other models though, with accuracy ranging from 62.1% to 92%.

Table 4 and Table 1 show how a model can show extremely good validation accuracy across all folds and appear to have fitted very well to the problem. However, when tested against the unseen data it has terrible accuracy. Having a small dataset here, was extremely detrimental to the learning of the model and it did not generalise well.

While the noise was originally added to the training data to increase robustness, the actual application does not have noise and just has a plain white background. The self-defined CNN model generalises well to recognising shapes for the specific user interface, or the specific scenario where the background is white. Using a different background, or attempting to use a photograph

of written shapes will not perform well with this model. The model with the highest accuracy here (k5, fold2) can be chosen and trained on more data.

The results of the training and validation process for the self-defined CNN and the pretrained resnet18 show how significant the training data for the model is. Since resnet18 has been trained on a large dataset, it is extremely optimised. It is difficult to train models on a small-scale dataset that a student will have the capabilities to gather.

6 Recommendations

For future iterations of this shape classification CNN, many things can be investigated to improve the model. Firstly increasing the size of the dataset that the model trains on is paramount. In addition to this, a more complex model could be used. The resnet18 model, which performed better, has 18 layers, compared to the 5 layers of my model. With the addition of layers, more computational power will be needed. Outsourcing the training to a computing system with more capabilities for machine learning would allow the model to be trained faster. This would allow for more iterations of hyperparameter tweaking. Research into preprocessing the images could be done, such as applying a median filter to remove the noise added.

7 Conclusion

Overall, the model has met the aim of being able to classify inputted shapes. The degree to which it can do this task differs, given varying circumstances.

The experimentation with different models, datasets, and training strategies sheds light on several aspects of shape classification. The performance disparity between the self-defined CNN and the pretrained resnet18 highlights the importance of training data scale and model optimization.

Resnet18, benefiting from its training on a large dataset, demonstrates superior performance across various conditions, including handling noisy data. Because it generalises well to unseen data, it can be leveraged for various applications. This is the concept of transfer learning, where an existing model is leveraged for your application, fine tuning it to meet the specific criteria.

The self-defined CNN struggles to cope with noise and exhibits minimal improvement to datasets containing added noise, indicating the challenges associated with training on a small-scale dataset.

While both models have their strengths and limitations, the results show the importance of dataset quality, model selection, and validation in achieving optimal performance in shape classification tasks.

References

[1] Convolutional neural networks for visual recognition. https://cs231n.github.io/classification/#:~:text=k%20%2D%20Nearest%20Neighbor%20Classifier,-You%20may%20have&text=The%20idea%20is%20very%20simple,recover%20the%20Nearest%20Neighbor%20classifier. Accessed: 2024-16-05.

[2] Nvidia cuda toolkit. https://developer.nvidia.com/cuda-toolkit. Accessed: 2024-16-05.

[3] Pytorch cifar10 tutorial. https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html. Accessed: 2024-16-05.

[4] Pytorch resnet18. https://pytorch.org/vision/main/models/generated/torchvision.models.resnet18.html. Accessed: 2024-16-05.

[5] Scikit api. https://scikit-learn.org/stable/modules/classes.html. Accessed: 2024-16-05.

[6] A. Antunes, B. Ferreira, N. Marques, N. Carrico. Hyperparameter optimization of a convolutional neural network model for pipe burst location in water distribution networks. *Journal of Imaging*, 9(3), 2023.

[7] A. Ghods, D. J. Cook. *A Survey of Techniques All Classifiers Can Learn from Deep Networks: Models, Optimizations, and Regularization.* PhD thesis, 2019.

[8] G. Hinton Y. LeCun, Y. Bengio. Deep learning. *Nature*, 521:436–44, 05 2015.

[9] L. Zhang. Hand-drawn sketch recognition with a double-channel convolutional neural network. *EURASIP Journal on Advances in Signal Processing*, 73, 2021.