Custom CNN Image Classification

Street view House Number (SVHN)

Matthew Alexander Prinsloo

1 Table of Contents

1. Image Classification

Data analysis is used to solve various tasks using different techniques and approaches. Supervised and unsupervised learning are the two main approaches to solving data analysis tasks. Each branch specialises in solving unique tasks using a particular set of data. Below is an overview of the two branches and the types of problems they are suited to solving:

Figure 1: Supervised vs Unsupervised Learning (Karan, 2024).

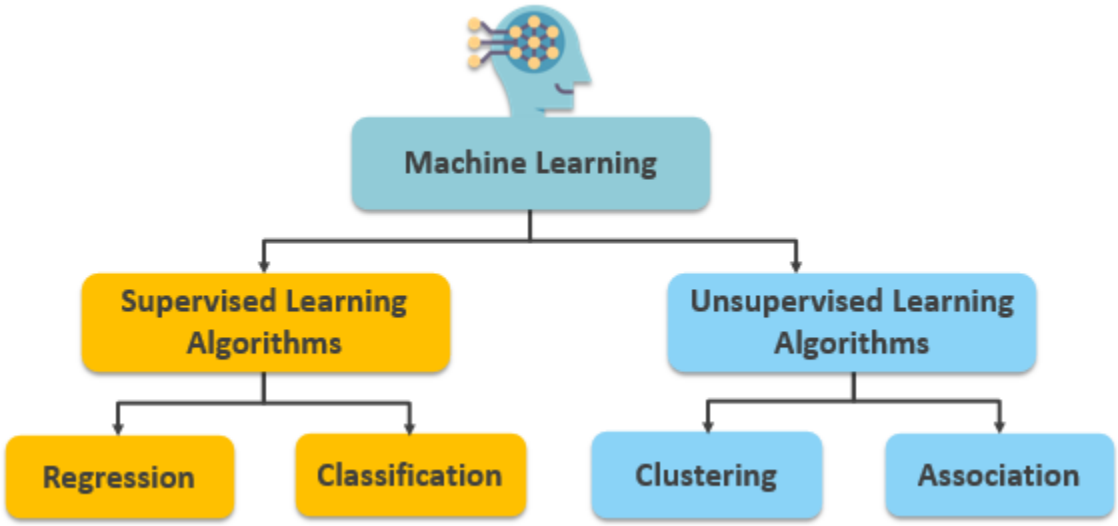


Image classification is one such task that involves categorising images into predefined classes (Banoula, 2024). Given that it is a type of classification, it falls under supervised learning as visualised in Figure 1. Unlike typical supervised learning tasks that use structured data, image classification combines both structured metadata and unstructured image data. Using unstructured data comes at a cost as it is more difficult to work with and requires attentive processing as it cannot fit within fixed data fields (Altexsoft, 2020). This suggests that along with attentive processing, one would need to consider what type of supervised learning technique is used with the image data.

While machine learning algorithms such as Random Forest, KNN, Decision Trees, and Naïve Bayes can be used for image classification, they tend to perform poorly (V, 2024). This was also observed in Task 2 of this portfolio of evidence where the student recorded poor performance for image classification when using logistic regression. Deep learning algorithms like convolutional neural networks (CNNs) are preferred for image classification tasks (V, 2024). This is likely due to the complexity of working with the unstructured nature of images.

Unlike in typical classification tasks where analysts will need to encode non-numerical data like ‘Male’ and ‘Female’ into numerical values like 0 and 1, images are more complicated. Similarly, for CNNs, it is common practice to normalise the pixel values of the images (V, 2024). Seeing that the pixels of images range from 0-255, simply dividing each pixel by 255 will result in each pixel ranging from 0-1, effectively being normalised. Once normalised, the pixels can then be fed to the input layer of the neural network in the form of 2D arrays (Biswal, 2024).

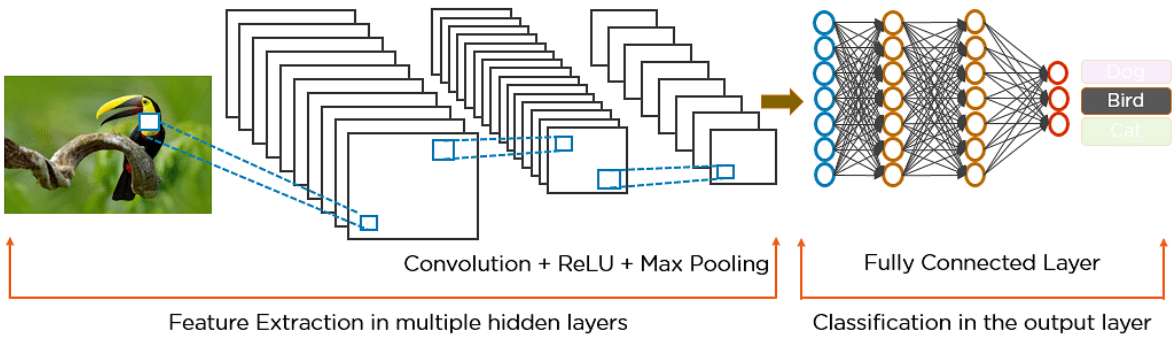
1. Image Classification

There are many layers to a CNN, here is an overview of the layers that can be found within them:

- Convolutional layer;
    - Consists of multiple linear and nonlinear operations to perform feature extraction;
    - The outputs of the linear function are fed to a nonlinear activation function such as ReLu.
- Pooling layer; and
    - Pooling is a downsampling operation that reduces the dimensionality of feature maps to allow for better generalisation in the face of small shifts and distortions. This reduction also influences the number of learnable parameters fed to the fully connected layer.
    - Max pooling is a form of pooling which extracts patches from the input feature maps and considers only the output of the max value of each patch while discarding the rest.
    - Global average pooling takes the average of all elements in each feature map and reduces it down to a 1x1 array whilst retaining the original depth.
- Fully connected layer (Yamashita, Nishio, Do & Togashi, 2018).
    - The output of the feature maps created from the last convolution or pooling layer is flattened into a one-dimensional array of numbers and fed to the fully connected layer.
    - Similar to the convolutional layer, each fully connected layer is followed by a nonlinear function such as ReLu. These layers are responsible for decision-making as each input is connected to every output by learnable weight.
    - The last layer of the fully connected layer has a different activation function compared to the rest, such as sigmoid or softmax. It also typically contains the same number of output nodes as the number of classes that could be predicted.

It is important to understand the basics of CNNs as they provide valuable insights into how one should select and process the data they collect to perform image classification. Below is a figure that showcases how an image of a bird would be used as input for a CNN model:

Figure 2: Functionality of CNN for image classification ***(Biswal, 2024)***.



1. Image Classification

Using the knowledge discussed in the overview, above we can note how the image was passed through the CNN and how it effectively predicted the image classification. First, the image was passed to a series of convolutional layers where the features were extracted and fed to nonlinear ReLu function. Max pooling was implemented to reduce the dimensionality while ensuring the learning process is more robust to slight shifts and distortions. Figure 2 does not mention it but it can be inferred that the data was flattened to a 1D array after pooling. This 1D array is then fed to the fully connected layer where multiclass classification is performed, likely using a softmax activation function in the final layer. When the information reaches the final layer, denoted in red with 3 nodes, the final decision is made for the classification of the image. The number of nodes here is equal to the number of classes that an image could be, Figure 2 shows that the CNN correctly identifies the image as a bird.

Image classification can be used in numerous computer vision scenarios given that the appropriate image data available is properly labelled. For instance, given that the data is available, one could train a model to classify if an image is of a dog or a cat or if a skin lesion is malignant or benign. In this task, the focus will be on classifying the central number of a street-view house number image.

2. Multiclass Classification

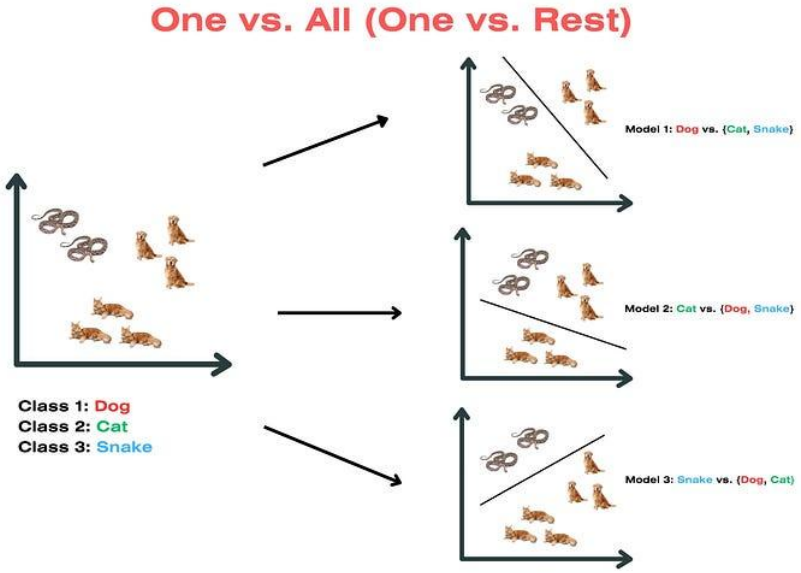
As discussed in Section 1 and visualised in Figure 1, classification is a branch of supervised learning. Classification can be further divided into two types:

1. Binary classification; and
    a. Involves a dataset with only two classes.
    b. Only requires one classifier model.
2. Multiclass classification (Murpani, 2023).
    a. Involves a dataset with more than two classes.
    b. The number of classifiers is dependent on the specific classification technique.

In this assignment, the student will utilise a convolutional neural network for multiclass image classification. This is because the dataset used represents more than two classes, ten classes to be exact. There are two different ways that multiclass classification can be applied with each approach requiring more than 1 classifier:

- One vs one – The number of classifiers is equal to the product of the number of classes and the number of classes minus one, divided by two, n(n-1)/2; and
- One vs all – The number of binary classifiers is equal to the number of classes (Murpani, 2023).

For this particular task, either approach could have worked but the decision was made to go with the one vs all as CNNs tend to perform better using this approach (Pawara, Okafor, Groefsema, He, Schomaker & Wiering, 2020). This means that the final layer of the fully connected layer will need to allocate 10 neurons for the output; one classifier for each of the possible classes. Below is a visualisation of how one vs all multiclass classification functions:



On the left is a sample of images that has three classes, ‘Dog’, ‘Cat’, and ‘snake’. Using the number of classes identified from the sample we can observe how three classifiers have been created to perform multiclass classification. This approach can be used in a variety of contexts including this assignment where the student aims to classify a street view house numbers focused digit as one of ten classes.

2. Multiclass Classification

The choice of activation functions for multiclass classification varies but most commonly, the output function uses softmax with the weights of the deep neural network optimised using the cross-entropy loss function (Pawara *et al.*, 2020). The sigmoid function could also be used here but the predictions would become uncertain as sigmoid does not have a range like softmax does. Softmax is likely used for this very reason, the classifier with the highest certainty for the class depicted in the image will be the final decision for the image classification.

The purpose of this analysis process is to provide a framework to classify street-view house images while using the recommended setup for CNN multiclass classification. In the scenario that door-to-door automatic delivery services become a service that is available to the public, a model that can classify a house number will be crucial in the performance of these systems. While geo-locating to the delivery location may suffice, honing in on the specific house number may be challenging without a robust image classifier for street-view house numbers. To ensure that the model is robust enough for potential deployment, the image classification model will need to have its performance properly evaluated. Common evaluation metrics for CNNs and by association, multiclass classification models are:

- Accuracy – The overall percentage of test images that the CNN correctly classified;
- Precision – It measures the percentage of test images that were correctly predicted as a specific class, it reveals the certainty of the model;
- Recall – Measures how well the model is at identifying a specific class, reveals the number of correctly identified classes; and
- F1 Score – The combination metric of precision and recall to provide an overview of how well the model performed (Biswal, 2024).

While there are other ways to evaluate the performance of the model, these metrics will serve as a good starting point for evaluating the performance of a multiclass image classification model. After performing the analysis, a well-developed, accurate and robust model capable of classifying the middle focus number present in a street-view house number image.

3. Dataset Used and Why it is Appropriate

In this task, the student identified and aimed to answer the following question: “Can one predict the class of a cropped street view house number image using a custom convolutional neural network?”. To answer this question, the student first needed to collect the appropriate data to analyse. The dataset used for the assignment was the Street View House Number (SVHN) dataset which contains Google Street View images of real-world house numbers. The specific configuration used for this task was the subset of SVHN called “cropped_digits”. As the name suggests, this subset is of the cropped full-house numbers with a single number being cropped to be the focus of an image. The student made use of both of the available “extra” splits which amount to more than 500,00 entries as shown during the EDA. Both the main page and the parquet file URL can be found below:

Dataset Link: https://huggingface.co/datasets/ufldl-stanford/svhn

Parquet files: https://huggingface.co/datasets/ufldl-stanford/svhn/tree/main/cropped_digits

The “cropped_digits” subset was chosen as it contains more entries and is better suited for simple multiclass image classification. This is because the model is only expected to predict the central/focus digit and not all digits present in an image. While the requirement for this task specifies the minimum entries to be 100,000, the student believed it beneficial to first consider a larger sample. By considering a larger sample, the student could explore and select the images of the highest quality while retaining a balanced distribution of classes. The goal is to have 10,000 high-quality entries for each of the 10 classes. Considering that the sample contains over 500,000 entries, this goal is feasible. Balanced data is important for ensuring generalisation, unbiased model training and trustworthy evaluation metrics because imbalanced data has been shown to fail in these aspects (Mazumder, 2024).

The schema of the parquet data files that were downloaded from huggingface.com had two columns:

- Image (binary); and
    - Image bytes – Hexadecimal values of encoded cropped images
    - Image path – null values
- label (long).
    - Whole numbers ranging from 0 to 9 to classify the image

When the bytes were converted and saved locally as PNG images, it was revealed that they were indeed RGB images of cropped house numbers with a resolution of 32x32 pixels. Having images at such a low resolution can be concerning as there is less for the model to consider during training. Seeing that the common resolution for image classification tasks is 224x224, upscaling the images to match this resolution will be easy as the images are already square and would not require any padding (Patel, 2023).

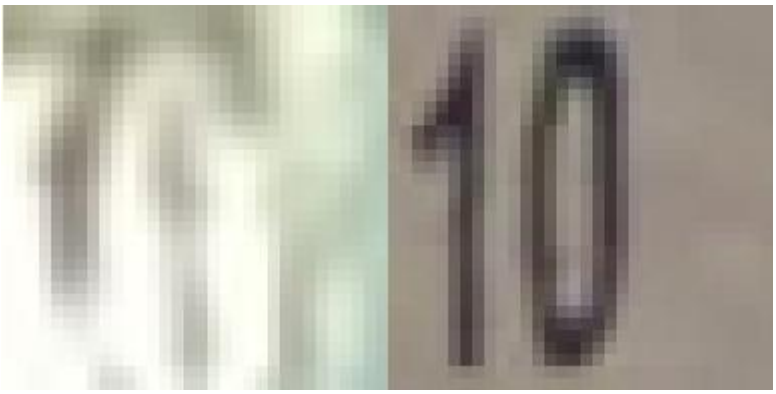
This dataset is appropriate for image classification as it fulfils the fundamental requirements for this task. These requirements are to:

1. Have sufficient and appropriate image data; and
2. Have labels for the image data.

3. Dataset Used and Why it is Appropriate

Seeing that this task will leverage a convolutional neural network (CNN), it is important to evaluate the weaknesses of this algorithm to ensure that the dataset collected is appropriate. CNNs have a couple of weaknesses but one of interest is that they have large data requirements (Biswal, 2024). Seeing that the minimum number of entries required for this assignment was met, this should be a null issue but, it is something to consider if the performance of the model is subpar. Thankfully, if the need arises to train the model using more data than the planned 100,000, the total data collected should be enough to combat the issue. While the student has access to over 500,000 entries, it may not be the best approach to train a model using all the images. This is because the student would prefer sampling images of higher quality from the total data collected. Below is a sample of the data collected which showcases how image quality can vary:

Figure 3: Sample of 32x32 source images labelled as zero.



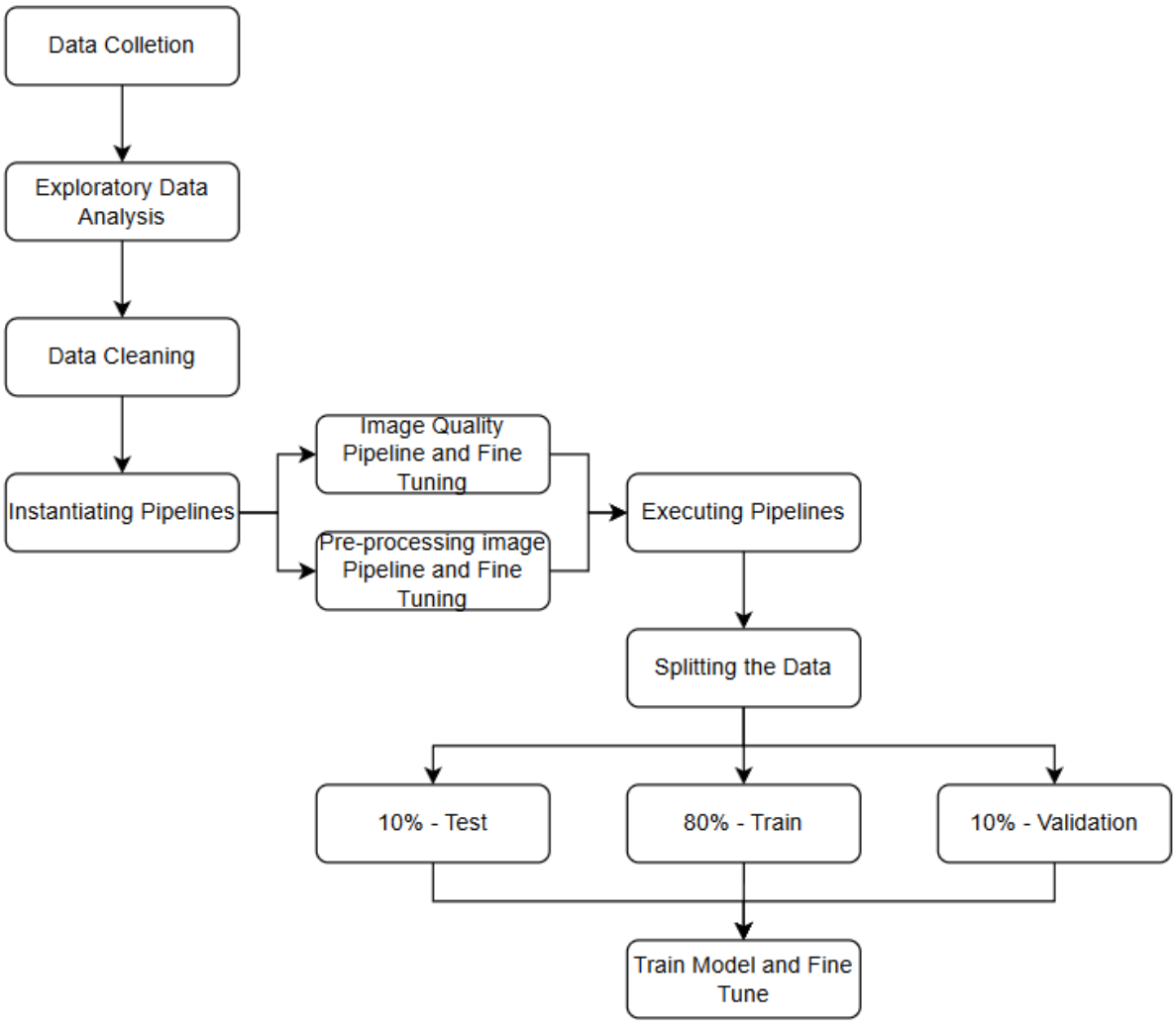
The image on the left is of far worse quality compared to the image on the right. It is important to consider the quality of the images fed to the CNN as it will influence how it will learn features. The CNN will have a harder time learning the features related to the number zero using the image on the left compared to the right. This is because of the quality of the sharpness, level of brightness, contrast and the presence of artefacts. The image on the right is still not perfect as it is still blurry and will need to be upscaled to 224x224. These concerns will be addressed in the code through upscaling and sharpening transformation. While poor-quality images are present, the dataset is still appropriate as it contains quality images that can be used to train a model. The student will need to filter through the images in the dataset to extract the images that are sharper and have a good balance of brightness and contrast with little to no artefacts.

4. Analysis Process

4.1. Overview

The focus of this section is to provide insight into the analysis process that was taken during the completion of this image classification task. The student will cover many aspects regarding the process however, the interpretation of the results will not be discussed here. This section will refer to the Jupyter notebook code document which should be available in the same directory as this Word document. To keep the discussion concise and manageable, the student will only discuss the results of the analysis process and keep the presentation of the actual code used to a minimum. An overview of the analysis process can be summarised using the provided flowchart below:

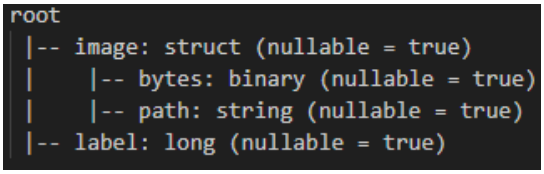
Figure 4: Overview of the analysis process.



4.2. Exploratory Data Analysis (EDA)

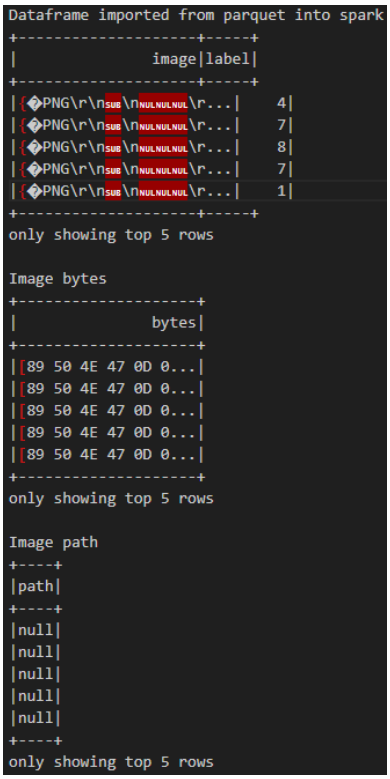
The analysis process begins with the exploration of the collected data in the Exploratory Data Analysis (EDA) section. In this section, the student gained some important insights into the nature of the SVHN dataset. The student was required to make use of spark throughout the data analysis process as far as possible. Once the two parquet files were read into a spark dataframe, the student was able to begin their exploration. The first thing the student wanted to check was the schema of the data they downloaded. Below is a snippet of the output of printing the schema from the spark:

Figure 5: Checking the schema of the freshly created spark dataframe.



The Schema already provided insight into how the student was going to approach the assignment. From the above output, we can see that there are two columns, a tuple named image that contains “bytes” and “path” and a second column named label. From here we can assume that “image” contains image data and the label contains numerical data for classifying the image. Since the student knew the background of the dataset they downloaded, they could assume that “label” would hold values from 0-9 but they were unsure of what would be contained in the image tuple. To remove the mystery the student decided to peek into each of the specific columns. Below is a snippet of the output showing the contents of the dataframe using Spark:

Figure 6: Showing the contents of the downloaded dataframe.



While reading the entire Dataframe is possible, as shown by the top output in Figure 6, the image data is not as clear as the label data is. From this output, one could observe that “label” contains plain whole numbers which shows that the data could be discreet but will need to be confirmed later. The middle output in Figure 6, “bytes”, alludes to the possibility that the images stored within the data frame have been encoded. For training purposes, these images will need to be decoded and saved locally for further examination. The image path appears to only contain null values which although strange, makes sense considering the file did not have any images inside of it. The student will check how many rows contain null values to see if their suspicion is true.

Figure 7: Output for checking the null values for every column in the dataframe.

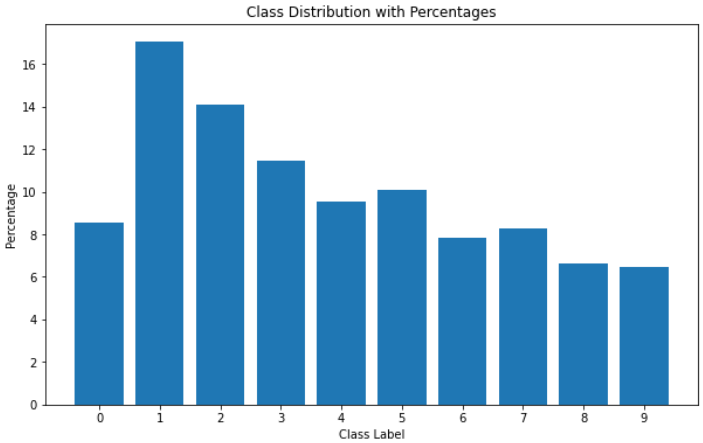
```
Total number of rows in the dataframe 531131
number of valid labels 531131
number of valid image bytes 531131
number of valid image paths 0
number of duplicate images 1
```

Here we can see that the dataset downloaded and used for the assignment meets the requirements of needing 100,000 entries as the total number of rows is equal to 531,131. Furthermore, we can see that it has been very well maintained as there are no nulls in the important columns. From this output, the student is comfortable dropping “images.path” and elevating “images.bytes” while dissolving the tuple structure. Interestingly, the student noticed that there was one duplicate image, this will also be dropped. Next, the student illustrated the distribution of the classes in the dataset. This allows the student to view the total classes and how they were distributed in the collected dataset.

Figure 8: Distribution of classes – illustrated using spark.

label	count	percentage
1	90560	17.050407526580074
2	74740	14.071857978540134
3	60765	11.440680359459341
5	53490	10.070961777791167
4	50633	9.533053050942234
0	45550	8.576038679723082
7	43997	8.283643771498934
6	41582	7.82895368562558
8	35358	6.657114723109742
9	34456	6.487288446729715

Figure 9: Distribution of classes – illustrated using a bar graph.



From the above figures, we can observe that the dataset, while not a horrible offender, is unbalanced to a degree. This unbalance will need to be monitored as it could influence the performance of the model created later. This output also confirms that there are 10 distinct classes, and they are represented properly in the dataset. If for instance, 11 classes appeared, the anomaly would be removed to clean to dataset. While the goal is to only use 100,000 images for this task, if the need arose to make use of more images, the student could at most use 34,456 images for each class. This is because the smallest class, '9', only has 34,456 entries. If the student wanted to employ under sampling to ensure class balance while using the most raw-images they had available, this is a solid approach to consider. This is just a contingency plan; the student hopes to only use the best quality images. With the dataset explored, the student was comfortable making the necessary decisions for creating a clean metadata dataset.

4.3. Data Cleaning

To clean the dataframe the student will need to perform the following tasks:

- Drop Duplicates;
- Drop images.path; and
- Elevate images.bytes and drop image.

The following is the new schema of the freshly cleaned dataframe:

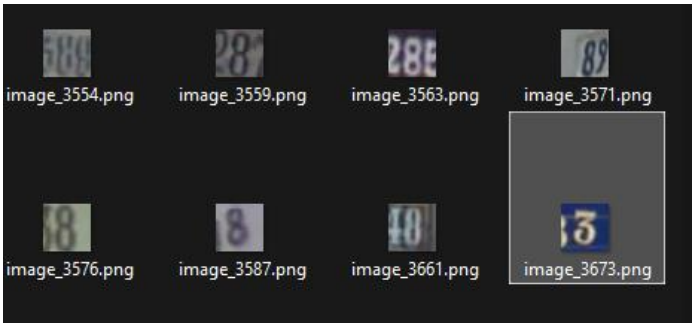
Figure 10: Schema of the cleaned dataframe.

```
root
 |-- label: long (nullable = true)
 |-- bytes: binary (nullable = true)

Number of entries: 531130
```

While the values indicate that the columns *could* contain null entries, the output shown in Figure 7 indicates that they do not contain any null values. For this reason, the student did not explicitly drop the nulls again during data cleaning as it was not necessary given the circumstances. With the data now properly cleaned, the student could then comfortably decode and download the images without fear of using unnecessary resources. Once the images were downloaded, the student was curious to explore whether the images were correctly labelled or not. Upon exploring the downloaded images, the student noted that for the most part, the majority of the images were correctly labelled with the rare occurrence of a misclassified image. This anomaly can be seen in the screenshot provided below:

Figure 11: Anomaly detected manually in the '8' class directory.



While this is a concern, the student is not too worried as the anomalies are a rare occurrence. Combatting this issue using the metadata alone is a difficult task as there are no supporting features other than the images that can confirm the classification of an entry. This issue could be resolved by:

1. Having a trained classification model sift through the dataset for anomalies; or
2. Manually identifying anomalies.

Option 1 was not feasible during the beginning of the project as the student did not have access to such as model. Option 2 could be done but considering the time constraints of this project, the student found this to be an unrealistic approach considering the frequency of the anomalies' appearance. With the metadata now sufficiently cleaned, the student could now turn their attention to pre-processing the images for splitting, training and evaluation. This will be handled in the next section where an image quality and pre-processing pipeline is created and executed.

4.4. Pipelines

This section will focus on describing the pipelines that were implemented during the analysis process. This section will detail the steps involved in the pipelines and how they were fine-tuned before being used during production.

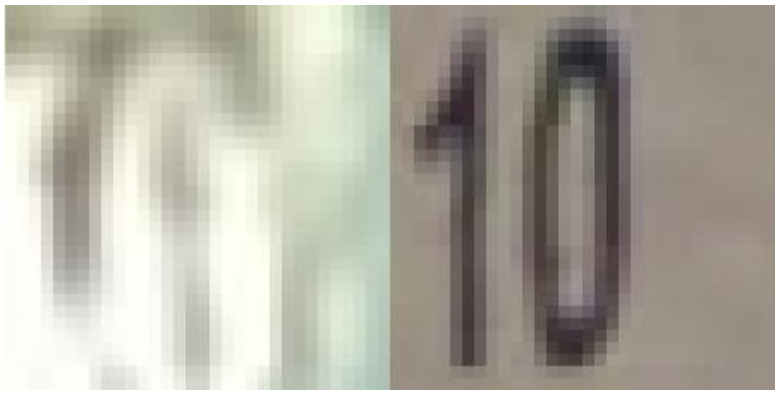
4.4.1. Parameter Tuning for Image Quality Assessment

The goal of this pipeline was to filter through the downloaded images and only select those that are of the best quality. Once the selection of best-quality images for a class reached 10,000 entries, the pipeline was instructed to stop filtering for that class and move on to the next. The following are the steps that were performed during the image quality pipeline:

1. Check if the image is blurry;
2. Check if the image has low contrast;
3. Check if the image is overly bright or dark; and
4. Check if the image has compression artefacts.

Fine-tuning the metrics for whether an image was of good or poor quality was done by altering the thresholds of the pipeline against manually chosen images. Below is a sample of what the student deemed as good and bad quality images:

Figure 12: Sample of 32x32 source images labelled as zero.



Once the majority of the good images passed and bad images failed, the thresholds were deemed to be well fine-tuned. The metrics below were the final thresholds defined by the student:

- blur_threshold = 140_000
- contrast_threshold = 17
- brightness_min = 50
- brightness_max = 200
- ssim_threshold = 0.95

4.4.2. Pre-processing Pipeline

This pipeline focussed on processing the images so they were ready to be fed to the CNN models. The steps taken during the pipeline are:

1. Ensure the image is RGB;
2. Resize the image with aspect ratio; and
3. Apply image sharpening;

Once the pipeline was set up the student only needed to focus on fine-tuning the sharpening of the images as that was the main adjustable variable. The student went through many iterations which can be viewed below:

Figure 13: The influence sharpening has on image quality.

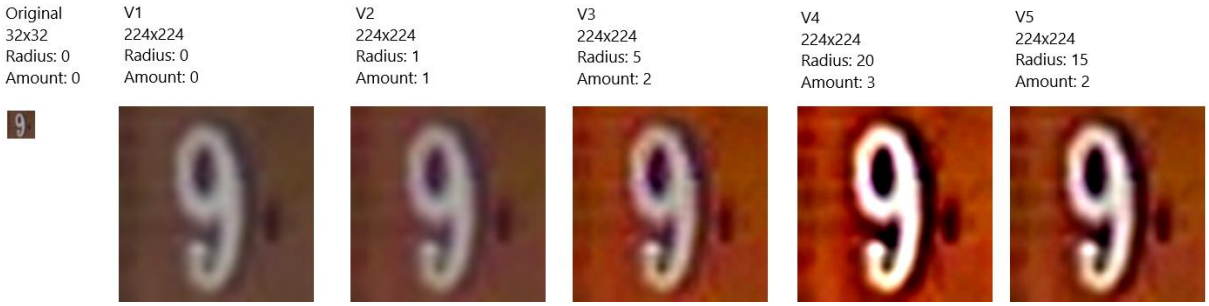


Table i: image quality metrics for fine-tuning sharpening.

Image 9	Blur	Contrast	Brightness	Compression
Original 32x32	243,855	28	82	0.956
V1	206	28	82	0.980
V2	229	28	82	0.971
V3	1,279	34	85	0.955
V4	32,547	65	88	0.918
V5	10,149	53	85	0.953

From the above Figure and Table, the student settled for version 5 of the fine-tuning process as it achieved the highest blur score (which means it is sharper) while ensuring that image compression remains over 95% unlike version 4. These metrics were only for image 9 but to find the results of the rest of the evaluated images please locate and open the folder labelled 'Fine_Tune_Sharpening_Docs'. This folder contains the text files that were produced during fine-tuning.

4.5. Splitting the Data

Here the student split the data that was processed by the previously defined pipelines. The two datasets created were ‘pre_process_0_0’ and ‘pre_processed_15_2’. These images needed to be split into 3 splits: train, test, and validate. Splitting the data was done using a standard 80/20 approach. Once the data was adequately split the following directories were created:

Table ii: Data splits created.

	Total Data	Train Split	Validation Split	Test Split
Split_0_0	10,000	80% 8,000	10% 2,000	10% 2,000
Split_15_2	10,000	80% 8,000	10% 2,000	10% 2,000

With the data adequately split, the student was now ready to begin training the CNN models.

4.6. Training the models

With the dataset sufficiently filtered, processed and split, the student was then ready to begin training the CNN models. The student made two attempts during the assignment with both attempts making use of the same CNN architecture but utilising different input data. Below is the architecture:

Figure 14: Custom CNN setup.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 222, 222, 32)      896

 max_pooling2d_3 (MaxPooling  (None, 111, 111, 32)     0
 2D)

 conv2d_4 (Conv2D)           (None, 109, 109, 64)      18496

 max_pooling2d_4 (MaxPooling  (None, 54, 54, 64)       0
 2D)

 conv2d_5 (Conv2D)           (None, 52, 52, 128)       73856

 max_pooling2d_5 (MaxPooling  (None, 26, 26, 128)      0
 2D)

 flatten_1 (Flatten)         (None, 86528)             0

 dense_3 (Dense)             (None, 128)               11075712

 dropout_1 (Dropout)         (None, 128)               0

 dense_4 (Dense)             (None, 64)                8256

 dense_5 (Dense)             (None, 10)                650

=================================================================
Total params: 11,177,866
Trainable params: 11,177,866
Non-trainable params: 0
_________________________________________________________________
```

The figure above shows how the CNN created has 3 convolutional layers which utilise the ReLu function as explored in the literature. It also makes use of Maxpooling before flattening and being fed to the fully connected layer. The fully connected layer has 2 hidden layers with the third hidden layer being the output layer. This is indicated in the above figure as it contains the same number of nodes as the number of classes (10). With the CNN model compiled, the student could now begin training, testing and evaluating the performance of the two attempts. Below is an overview of the configuration used for each attempt during this assignment:

Table iii: Models created during the analysis process.

	Attempt 1	**Attempt 2**
Model Used	Custom CNN	Custom CNN
Model Name	CustomCNN_0_0	CustomCNN_15_2
Dataset Used	Balanced class data	Balanced class data
Images Used	32 x 32 RGB images Upscaled to 224 x 224	32 x 32 RGB images Upscaled to 224 x 224 and sharp
Sharpened Radius	0	15
Sharpened Amount	0	2
Epochs	10	10

One can note that the major difference between the two attempts is that the first makes use of upscaled unsharpened data while the other makes use of upscaled sharpened data. The student at this point assumed that Attempt 1 should perform adequately but Attempt 2 should still perform better. In the next section, the student will explore and interpret the results of training these two custom CNN models.

5 Interpretation of Results

5.1 Attempt 1 – Upscaled Images

5.1.1 Attempt 1 – Native Windows Environment

When trained on the upscaled image data, the model was tasked to save the best iteration of the model at the lowest validation loss recorded. During training, the best model was recorded at epoch 10 which could be indicative of underfitting. This is because the model might require more epochs to extract the information necessary to conduct image classification. Using Tensorbaord, the student visualised the following two figures which describe the performance of the model over 10 epochs:

Figure 15: Accuracy of attempt 1 over 10 epochs.

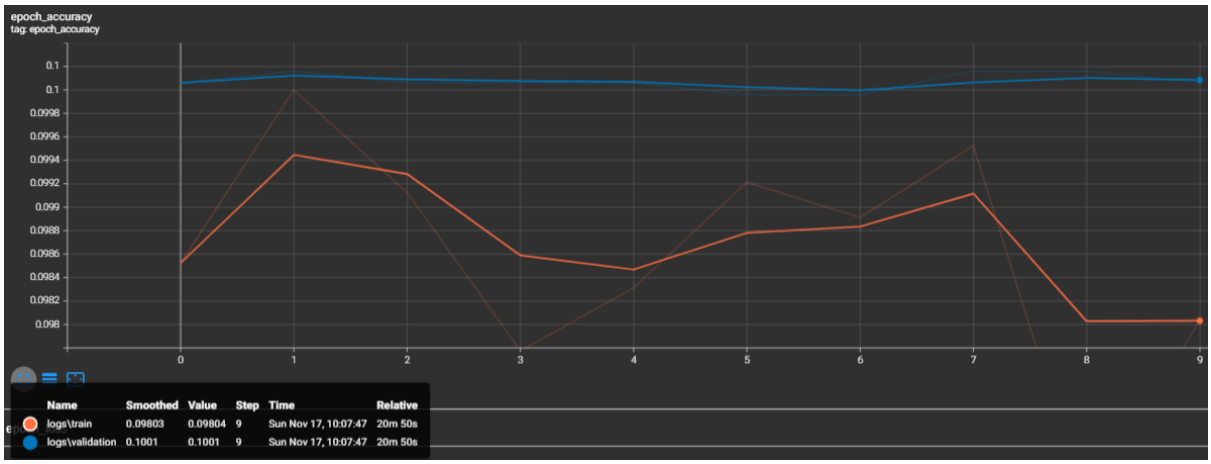
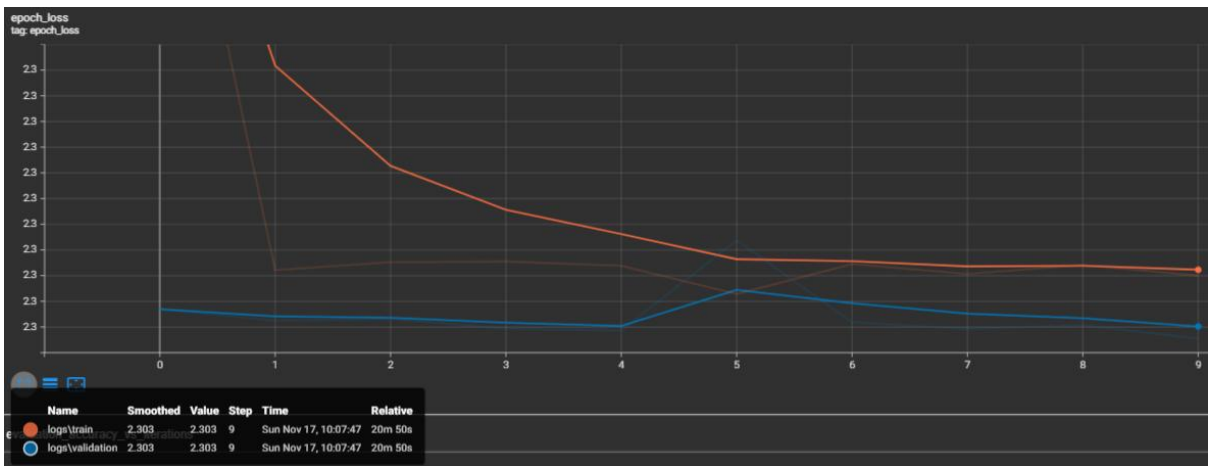


Figure 16: Loss of attempt 1 over 10 epochs.



From the above two figures, we can see that the model is performing horribly. At the best epoch for loss, the accuracy was only 1%. This shows that the model has not grasped the foundations for most of the features present in the 10 classes for this task. The student will need to check whether this is the case for all classes or just a handful. This is important to check for multiclass classification as some classes that perform poorly can skew the performance metrics for the data. Below are two figures which showcase the performance of the model across the 10 classes on the unseen test data split:

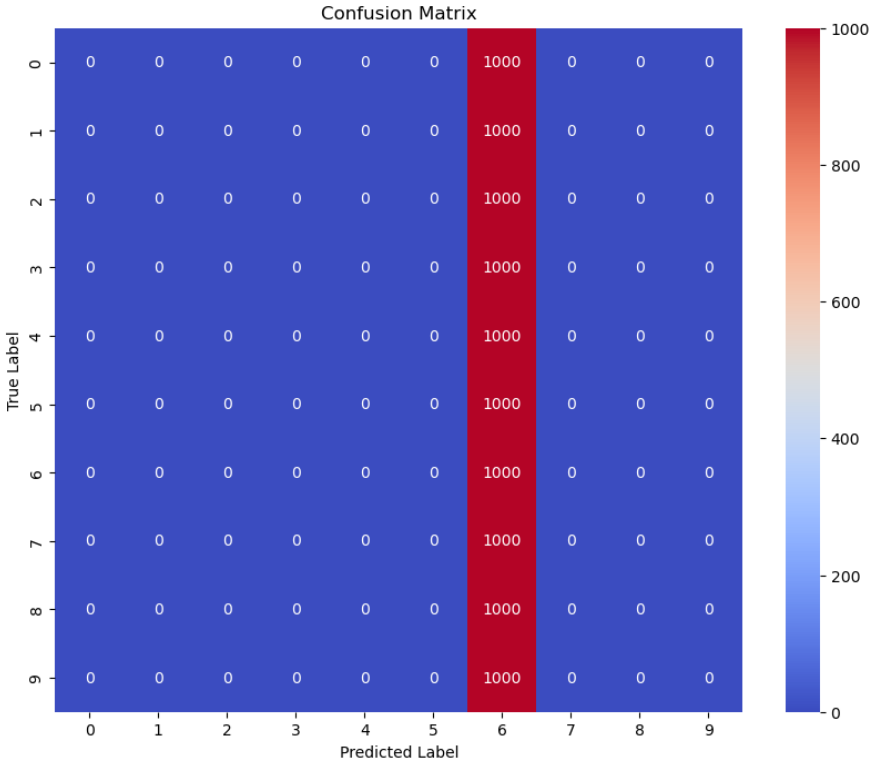
Figure 17: Classification report for attempt 1 – Native Windows.

```
Found 10000 images belonging to 10 classes.
313/313 [==============================] - 52s 166ms/step - loss: 2.3026 - accuracy: 0.1000
Test Loss: 2.302628755569458
Test Accuracy: 0.10000000149011612
313/313 [==============================] - 15s 47ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      1000
           1       0.00      0.00      0.00      1000
           2       0.00      0.00      0.00      1000
           3       0.00      0.00      0.00      1000
           4       0.00      0.00      0.00      1000
           5       0.00      0.00      0.00      1000
           6       0.10      1.00      0.18      1000
           7       0.00      0.00      0.00      1000
           8       0.00      0.00      0.00      1000
           9       0.00      0.00      0.00      1000

    accuracy                           0.10     10000
   macro avg       0.01      0.10      0.02     10000
weighted avg       0.01      0.10      0.02     10000
```

Figure 18: Confusion matrix for attempt 1 – Native Windows.



The classification report shows that all classes besides class 6 are performing poorly. Unfortunately, this is the worst-case scenario as class 6 shows perfect recall which is indicative that it is simply predicting every class to be class 6. The confusion matrix above shows that the model predicts every value provided to it as the number 6. This explains the perfect recall shown in the classification report. Typically, this would be a classic indicator that the dataset is imbalanced but this cannot be the case as the data used during training was perfectly balanced.

5. Interpretation of Results

Overall this model performed poorly on both the validation and test set which indicates it is severely underfitted. This could be due to the lack of input data but this is unlikely as it should not be this underfitted. Seeing that the images were not sharpened, the student at this stage assumed that this could have been the main cause of the performance. ‘Garbage in garbage out’, this mantra came to mind at this point during the analysis as even the logistic regression model performed better on this data compared to the custom CNN. The difference between task 2 and this one is that the images were upscaled which could have degraded the quality. While an assumption at the time, the student confirmed their suspicions when fine-tuning the sharpening metrics. This can be seen in Table i: image quality metrics for fine-tuning sharpening.

5.1.2 Attempt 1 – WSL2 Environment

This project was revisited a year later with a different training environment, Windows Subsystem for Linux (WSL). Following the same prior evaluation and testing methodologies, the following metrics were documented:

Figure 19: Accuracy of attempt 1 over 10 epochs - WSL2.

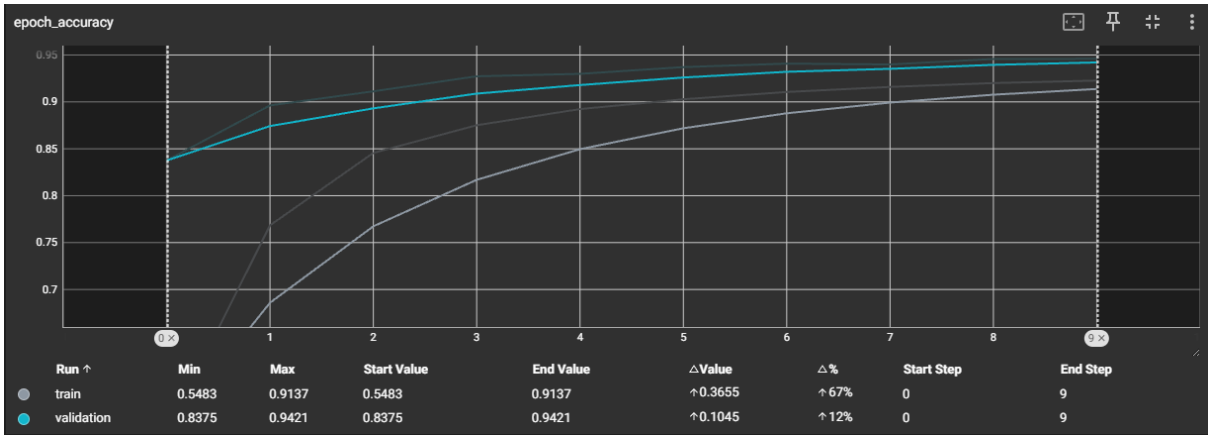
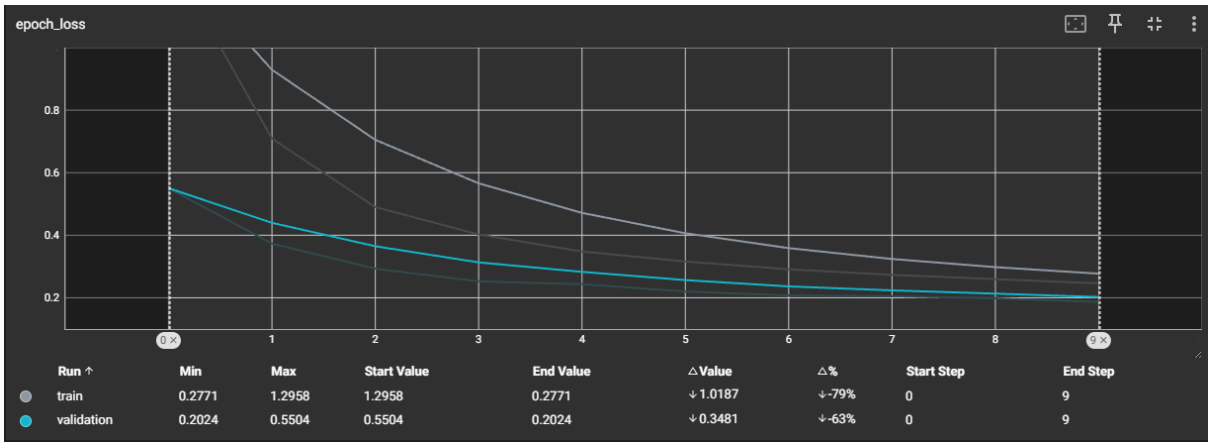


Figure 20: Loss of attempt 1 over 10 epochs - WSL2.



From the above two figures, we can see that altering the environment has dramatically influenced the performance of the custom CNN model. The model is not only learning but it is doing so incredibly well. The model has gone from having an accuracy of 1% to 94%. This model was only trained to the tenth epoch for testing purposes but going forward it would be interesting to see if there would be convergence between the training and validation curves. The best model was recorded at the tenth epoch so perhaps stopping here and not going further is the right call as no convergence means the chances of overfitting are slimmer even if there is more performance to eek out.

Both the accuracy and loss diagrams do not show any signs of exploding or vanishing gradient. There are also no signs of over or underfitting as the curves remain at a healthy distance from one another. We can look to the confusion matrix and the classification report to confirm that the model is generalising well to unseen data across all classes or not.

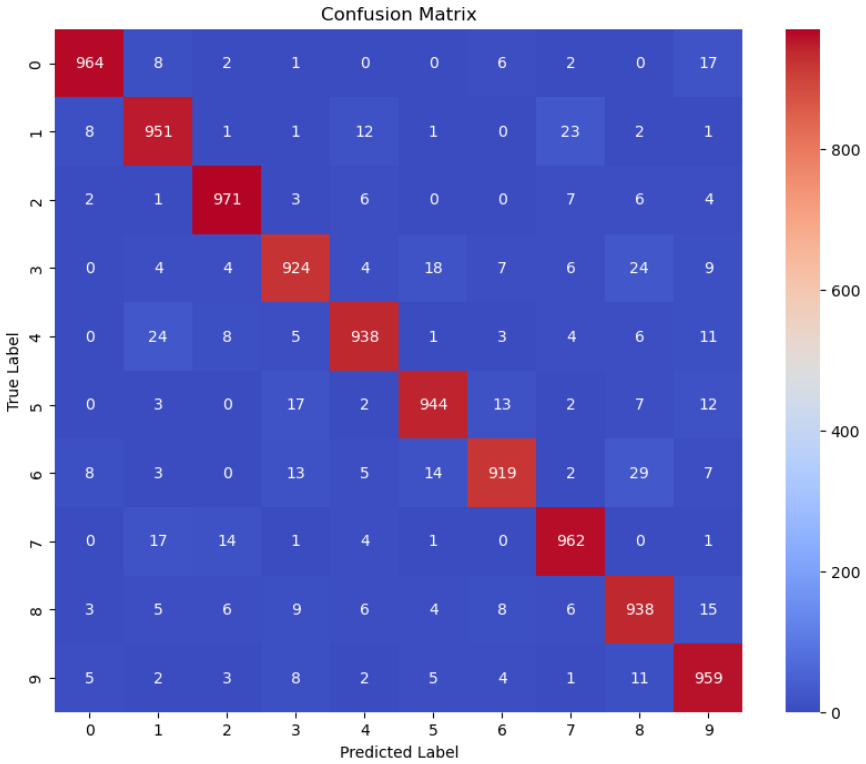
Figure 21: Classification report for attempt 1 – WSL2.

```
Found 10000 images belonging to 10 classes.
313/313 ━━━━━━━━━━━━━━━━━━━━ 21s 64ms/step - accuracy: 0.9470 - loss: 0.1745
Test Loss: 0.17454248666763306
Test Accuracy: 0.9470000267028809
313/313 ━━━━━━━━━━━━━━━━━━━━ 15s 46ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.96      0.97      1000
           1       0.93      0.95      0.94      1000
           2       0.96      0.97      0.97      1000
           3       0.94      0.92      0.93      1000
           4       0.96      0.94      0.95      1000
           5       0.96      0.94      0.95      1000
           6       0.96      0.92      0.94      1000
           7       0.95      0.96      0.95      1000
           8       0.92      0.94      0.93      1000
           9       0.93      0.96      0.94      1000

    accuracy                           0.95     10000
   macro avg       0.95      0.95      0.95     10000
weighted avg       0.95      0.95      0.95     10000
```

Figure 22: Correlation Matrix for Attempt 1 – WSL2.



These two outputs confirm that the model is not overfitted to the training data as it is able to correctly predict the class for the numbers in the validation set with incredible accuracy. The model appears to struggle more with predicting 3 and 8. This is understandable as the two numbers are similar in shape, especially because the borders of the image are may not be as well defined without using image sharpening. It'd be interesting to see if these issues are mitigated in Attempt 2 as the theme between 3 and 8 extend into other numbers as well like 1 , 7 and 4.

5.2 Attempt 2 – Upscaled and Sharpened

5.2.1 Attempt 2 – Native Windows Environment

When trained on the upscaled and sharpened image data, the model was tasked to save the best iteration of the model at the lowest validation loss recorded. During training, the best model was recorded at epoch 4. Stopping at epoch 4 may be a cause for concern considering that it could train for 10 epochs but on the positive note, the model does not have a long training time. Using Tensorbaord, the student visualised the following two figures which describe the performance of the model over 10 epochs:

Figure 23: Accuracy of attempt 2 over 10 epochs.

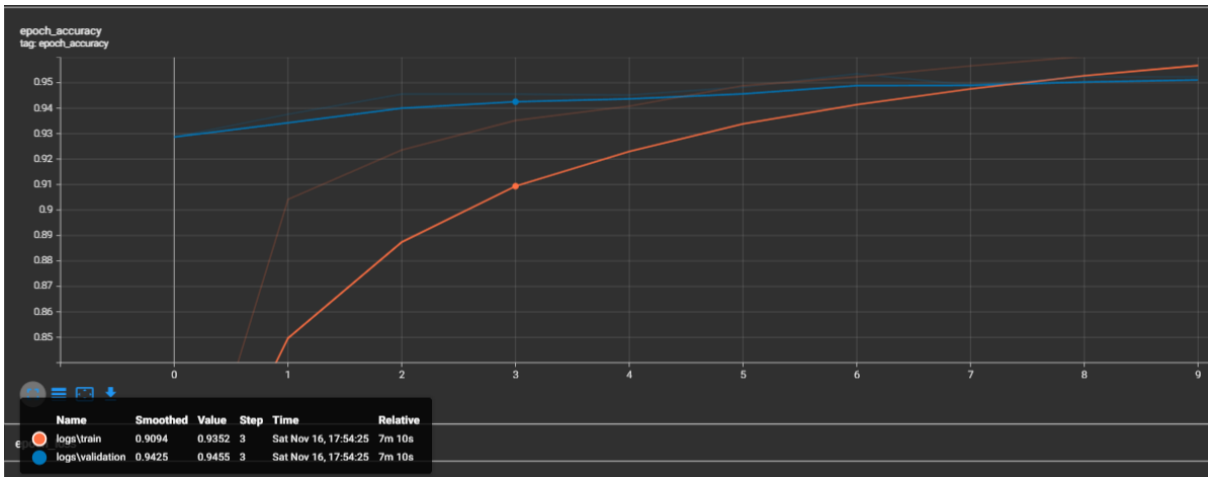
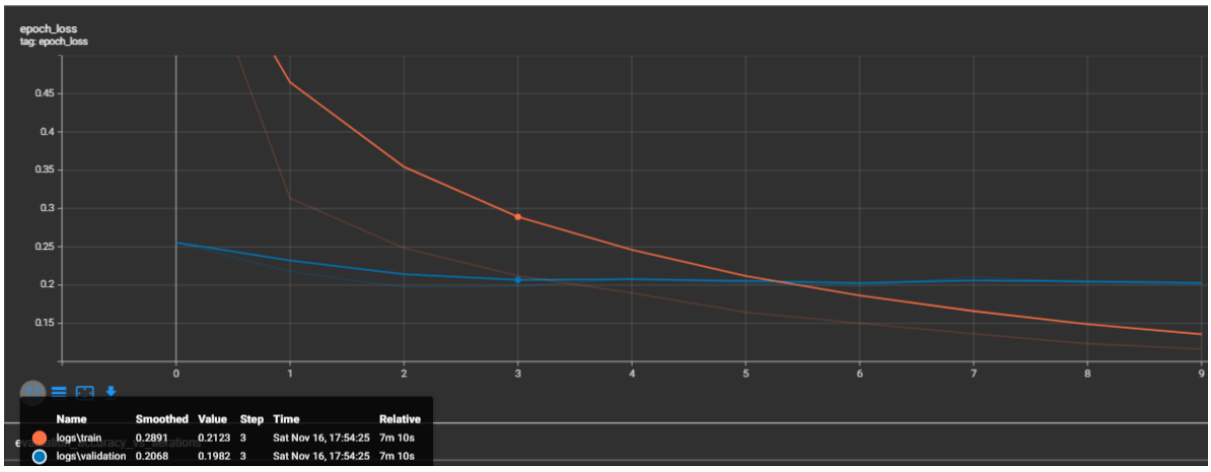


Figure 24: Loss of attempt 2 over 10 epochs.



In the lighter colours, we can see the raw/unsmoothed representation of the loss over the 10 epochs. We can see that the loss recorded at epoch 4 is where the unsmoothed train and validation loss converged. After convergence, the model's validation loss plateaued until the last epoch was reached. This confirms that it was indeed the right call to stop training at epoch three as the validation loss did not improve after then. Training loss did improve but this is not as important as the model's ability to predict semi-unseen data. The accuracy of the model also began to plateau at epoch 4 but climbed slowly afterwards, stopping here was the right call to prevent potential overfitting. Regardless though, the model is performing exceptionally well in terms of both accuracy and loss. The accuracy recorded for the best iteration of the

model is roughly 94% and the loss is equal to 0.2 for the validation set. This is quite concerning as it could mean that the model has been overfitted. To confirm or deny this suspicion, the student has evaluated the model on the unseen test split to see if it can effectively generalise.

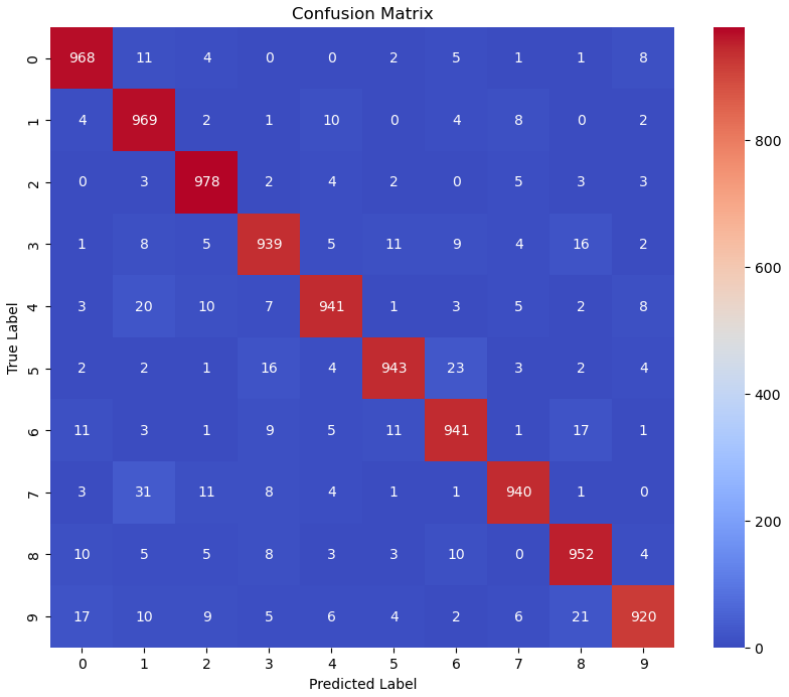
Figure 25: Classification report for attempt 2.

```
Found 10000 images belonging to 10 classes.
313/313 [==============================] - 59s 169ms/step - loss: 0.1831 - accuracy: 0.9491
Test Loss: 0.183111310005188
Test Accuracy: 0.9491000175476074
313/313 [==============================] - 16s 52ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.97      0.96      1000
           1       0.91      0.97      0.94      1000
           2       0.95      0.98      0.97      1000
           3       0.94      0.94      0.94      1000
           4       0.96      0.94      0.95      1000
           5       0.96      0.94      0.95      1000
           6       0.94      0.94      0.94      1000
           7       0.97      0.94      0.95      1000
           8       0.94      0.95      0.94      1000
           9       0.97      0.92      0.94      1000

    accuracy                           0.95     10000
   macro avg       0.95      0.95      0.95     10000
weighted avg       0.95      0.95      0.95     10000
```

Figure 26: Confusion matrix for attempt 2.



5. Interpretation of Results

The model is performing extremely well on unseen data, with an average test accuracy of roughly 94%. The concern that the model was overfitted can now be dismissed as it has successfully generalised to completely unseen test data. The performance of this model shows how important it is to enhance the quality of the input images for a CNN. Changing one step in how the model was trained produced a model that is performing outstandingly well compared to a model where guessing would yield better results. This is likely due to the results shown in Table i: image quality metrics for fine-tuning sharpening., there is a vast difference in sharpness in V1 (used in attempt 1) and V5 (used in attempt 2). This difference in sharpness explains why attempt 1 was underfitted as the blur degraded the quality of the features extracted in each image. For this task, it appears that it is crucial to ensure that the hardlines of the number in the street view house number images are important for proper feature extraction.

The confusion matrix shows that while the model is performing well, there are areas where it has still predicted the class incorrectly. These incorrect predictions are minimal though which shows that the model has successfully generalised to an unseen population. While this model is performing well, future steps could be taken to further improve it. These steps could be:

- Fine-tune sharpening further;
- Leverage transfer learning;
- Introduce more data; and
- Fine-tune the CNN using grid search.

5.2.2 Attempt 2 – WSL2 Environment

Unlike the native windows environment where the best model was trained at the 7th Epoch. While it took longer to reach the same level of accuracy as the native windows environment, the new configuration appears to be performing ever-so-slightly better. This can be observed by analysing the following two graph relating to accuracy and learning loss.

Figure 27: Accuracy of attempt 2 over 10 epochs - WSL2.

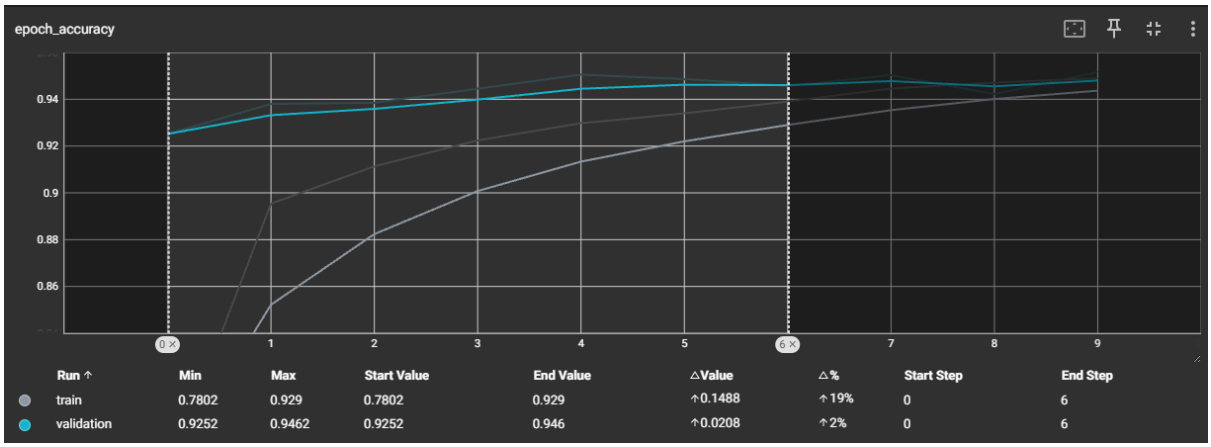
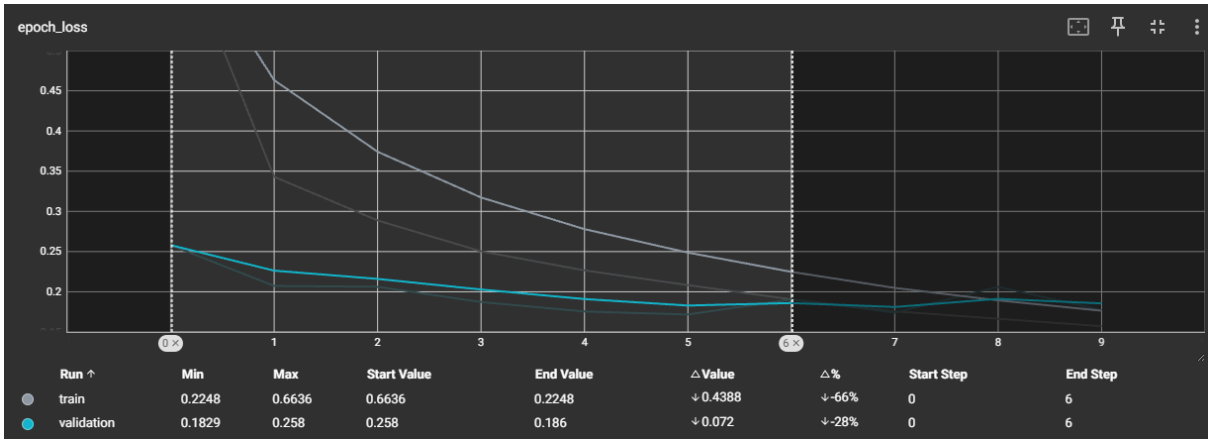


Figure 28: Loss of attempt 2 over 10 epochs - WSL2.



Convergence only took place in the loss graph at the Epoch 6 for the unsmoothed data. While the performance continued to improve for the training set, the model had begun to plateau relatively quickly for the validation set. The model was automatically stopped here as there was no improvement over the next two epoch from here. As always with a well performing model, one needs to dispel the suspicion of the model being overfitted by reviewing the confusion matrix and the classification report.

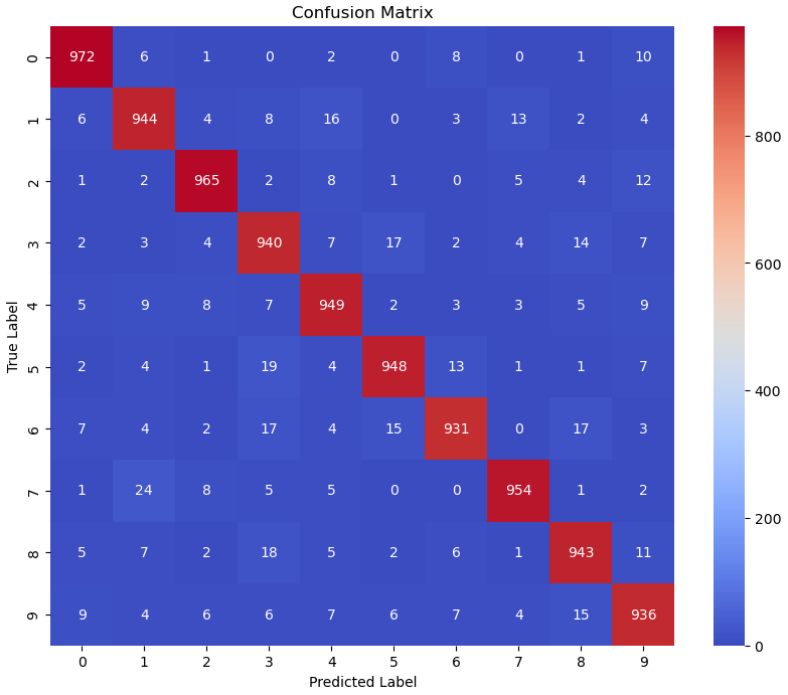
Figure 29: Classification report for attempt 2 – WSL2.

```
Found 10000 images belonging to 10 classes.
313/313 ━━━━━━━━━━━━━━━━━━━━ 23s 72ms/step - accuracy: 0.9482 - loss: 0.1764
Test Loss: 0.17642195522785187
Test Accuracy: 0.948199987411499
313/313 ━━━━━━━━━━━━━━━━━━━━ 16s 51ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.97      0.97      1000
           1       0.94      0.94      0.94      1000
           2       0.96      0.96      0.96      1000
           3       0.92      0.94      0.93      1000
           4       0.94      0.95      0.95      1000
           5       0.96      0.95      0.95      1000
           6       0.96      0.93      0.94      1000
           7       0.97      0.95      0.96      1000
           8       0.94      0.94      0.94      1000
           9       0.94      0.94      0.94      1000

    accuracy                           0.95     10000
   macro avg       0.95      0.95      0.95     10000
weighted avg       0.95      0.95      0.95     10000
```

Figure 30: Correlation Matrix for Attempt 2 – WSL2.



The model performed similarly as before in the native windows environment; it has an average test accuracy of 0.948 as opposed to 0.949 from before. While performing slightly worse, the model has seen some improvement with regards to the problem numbers:

Table iv: Redistribution of predicted classes for problem number

Prediction Vs. Answer	Before	After
1-7	31	24
7-1	8	13
1-4	20	9
4-1	5	16
8-3	16	14
3-8	8	18

We notice that there is a trad-off between how it performed previously as opposed to now, the problem variables have seen an improvement in accuracy compared to last time. A pattern emerges where a variables prediction was bad and now is good and vice versa. The performance between the two models will be negligible however.

6. Concluding the Project

6.1 Conclusion & Future Directions

In conclusion, the task was a success as the student was able to answer the question “Can one predict the class of a cropped street view house number image using a custom convolutional neural network?”. The answer to this question is yes, it is possible. With the new WSL2 environment, both Attempts performed similarly on unseen data with Attempt 2 performing slightly better at 94.8% compared to 94.7% accuracy. While the author suspected that the accuracy may improve for problem classes, the results found that they performed similarly. Below is a table showing the number of times the model incorrectly categorized a class for attempts one and two:

Table v: Predicted classes for problem number – Attempt 1 Vs. Attempt 2

Prediction Vs. Answer	Attempt 1 – WSL2	Attempt 2 – WSL2
1-7	17	24
7-1	23	13
1-4	24	9
4-1	12	16
8-3	24	14
3-8	9	18

While the two models performed similarly, the author suspects that the performance may diverge when tasked to predict lower quality images. Going forward with future iterations of this project, the author will extract a subset of the SVHN images to evaluate the model’s performance on unseen lower quality images. This experiment will need to be done with care as ‘Low Quality’ will first need to be quantified to best understand how robust our models are.

The author originally had a GUI available for manual testing but it was bare bones and did not make use of any of the modern standards for python GUI design. This gap will be addressed by creating an application using the PyQt Python library to manually predict images provided by a user. This project also has the potential to be extended into a full Computer Vision application where one could use a live feed (OpenCV) to segment an image and predict the numbers identified. For practicality the author suggests using an existing image segmentation models like Segment Anything Model (SAM), YOLO's segmentation capabilities, DeepLabV3, etc..

6.2 Code Availability Statement:

Code File Provided:

You can find the Jupyter Notebook file provided in the same directory as this Word Document. It should be labelled as “MAPrinsloo_SVHN-CNN.ipynb”

GPU Enabled Environment:

enironment.tf-gpu-wsl.yml

GitHub:

https://github.com/MAPrinsloo/SVHN_CNN_PDAN8412-POE-S2P3

Models Created:

You will find the best models created and saved in the following directories:

- CustomCNN_model_0_0/checkpoints
  - ‘best_model_0_0.h5’
- CustomCNN_model_15_2/checkpoints
  - ‘best_model_15_2.h5’

Tensorboard Logs:

The most important logs have been saved to the Jupyter notebook and this document but for those who would like to interact with them directly in tensorboard may do so using the logs saved in the following directories:

- CustomCNN_model_0_0/logs
- CustomCNN_model_15_2/logs

The author acknowledges that their may be other ways to access tensorboard to view the files but the following instructions describe the way they did it using Anaconda Navigator.

Step 1: Locate the directory of the logs, for instance “E:\PDAN8412 POE PART 3\CustomCNN_model_15_2\logs”.

Step 2: Open “Anaconda Navigator”

Step 3: Open “CMD.exe Prompt”

Step 3: In the terminal, use the following command, ensuring you insert your path in place of ‘your_path_to_dir’:

tensorboard--logdir="your_path_to_dir\CustomCNN_model_0_0\logs"

Step 4: If successful, the terminal should output a link that locally hosts the tensorboard. Clicking on the link should redirect and open a browser tab to view the performance of the specified model.

7 References

Altexsoft. 2020. *Structured vs Unstructured Data: What is the Difference?* Available: https://www.altexsoft.com/blog/structured-unstructured-data/ [2024, August 31].

Banoula, M. 2024. *Supervised and Unsupervised Learning in Machine Learning*. Available: https://www.simplilearn.com/tutorials/machine-learning-tutorial/supervised-and-unsupervised-learning [2024, November 17].

Biswal, A. 2024. *Convolutional Neural Network Tutorial*. Available: https://www.simplilearn.com/tutorials/deep-learning-tutorial/convolutional-neural-network [2024, November 17].

Karan, R. 2024. *Differences Between Supervised and Unsupervised Learning - Shiksha Online*. Available: https://www.shiksha.com/online-courses/articles/differences-between-supervised-and-unsupervised-learning/ [2024, November 17].

Mazumder, S. 2024. *How to Handle Imbalanced Data?* Available: https://www.analyticsvidhya.com/blog/2021/06/5-techniques-to-handle-imbalanced-data-for-a-classification-problem/ [2024, November 18].

Murpani, R. 2023. *A Comprehensive Guide to Multiclass Classification in Machine Learning | by Ronit Murpani | Medium*. Available: https://medium.com/@murpanironit/a-comprehensive-guide-to-multiclass-classification-in-machine-learning-c4f893e8161d [2024, October 14].

OpenAI. 2024. *Chat-GPT (GPT-4o) [Large Language Model]*. Available: https://chatgpt.com/share/673bb312-cf9c-8008-981e-327d1fccac8e [2024, November 18].

Patel, M. 2023. *The Complete Guide to Image Preprocessing Techniques in Python | by Maahi Patel | Medium*. Available: https://medium.com/@maahip1304/the-complete-guide-to-image-preprocessing-techniques-in-python-dca30804550c [2024, November 17].

Pawara, P., Okafor, E., Groefsema, M., He, S., Schomaker, L.R.B. & Wiering, M.A. 2020. One-vs-One classification for deep neural networks. *Pattern Recognition*. 108:107528. DOI: 10.1016/J.PATCOG.2020.107528.

6. References

V, N. 2024. *Image Classification using Machine Learning - Analytics Vidhya*. Available: https://www.analyticsvidhya.com/blog/2022/01/image-classification-using-machine-learning/ [2024, November 17].

Yamashita, R., Nishio, M., Do, R.K.G. & Togashi, K. 2018. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*. 9(4):611–629. DOI: 10.1007/S13244-018-0639-9/FIGURES/15.

8 Disclosure of AI Usage in My Assignment

Sections:

All sections, cells and paragraphs utilised AI, this is due to the nature of the prompt given to Chat-GPT. It was tasked with reviewing the entire document, marking and providing feedback. Additionally, it would also check to see if anything was missing from the document. Thankfully, a passing grade was given from Chat-GPT to the student. The student did not make any alterations to the document bar this section and the reference list which could only be completed once the review was complete. Grammarly is an assistive writing tool that was used for the full duration of the writing of this document. This aids in correcting spelling and grammatical errors.

Name of AI Tool(s) used:

Chat-GPT (GPT-4o)

Grammarly

Purpose/Intention:

To ensure that the student has not missed anything and to provide a level of reassurance that they haven't failed any sections. It is also used to provide instantaneous feedback which will be useful for the student to understand before meeting with their lecturer again.

Date(s) in which generative AI was used:

18-November-2024

Link:

https://chatgpt.com/share/673bb312-cf9c-8008-981e-327d1fccac8e

(OpenAI, 2024)