Machine Learning Engineer Nanodegree

Capstone Project:

Inventory Monitoring at Distribution Centres

Fatima Akhter
February 2022

Project Overview:

This project developed a model that counts the number of objects seen in an image. The use case comes from distribution centres that use robots to move objects as part of their operations. Many objects are usually put together for delivery, there is a need to count the number of items so that deliveries can be as accurate as possible as items are not missed. A object counting model that can be integrated to do this will help companies with greater efficiencies and cost reduction.

Problem Statement:

The goal of this project is to create an object counting model built on AWS Sagemaker.
The task involved are the following:

1. Download and process the data , getting it ready for modelling
2. Training a model to recognise the number of objects
3. Evaluating the performance of the model

The Dataset:

The dataset (as explained in the proposal) contains five categories pertaining to the number of classes describing the number of objects seen in a dataset. In the remainder of the report Class 1 would mean the category containing 1 object and so on.

The following was explained:

To complete this project we will be using the Amazon Bin Image Dataset[1]. The dataset contains 500,000 images of bins containing one or more objects. We have been provided with a subset of this data. Within this there are 5 classes referring to the number of objects in each image. For each image there is a metadata file containing information about the image like the number of objects, it's dimension and the type of object. The subset of the dataset provided has imbalanced classes, the number of objects being:

Class	*Number of images*
1	*1228*
2	*2299*

[1] https://registry.opendata.aws/amazon-bin-imagery.

3	*2666*
4	*2373*
5	*1875*

Metrics:

Since the classes are highly imbalanced it is imperative we look into the within class metrics for accuracy, precision and recall.

Accuracy: For each class how many are correct

$$Accuracy = \frac{true\ positives + true\ negative}{class\ size}$$

Precision: what proportion of the predicted class is the actual class

$$Precision = \frac{true\ positive}{true\ \ positive + false\ positive}$$

Recall: what proportion of the actual class is predicted correctly

$$Recall = \frac{true\ positive}{true\ positive + false\ negative}$$

All of these are important, the model may be biased for a class where is constantly predicts a certain class, or the model may consistently confuse one class for another. This can help understand where the model will need to do better or if the imbalanced nature of the dataset means the model may not have enough examples . We will need to use these metrics to recommend further work.

Analysis

Data Exploration

To understand the complexity of the problem we had a look at some images in the dataset. The following show three images from class 1,2,3 respectively

Fig 1: Example image from class 1

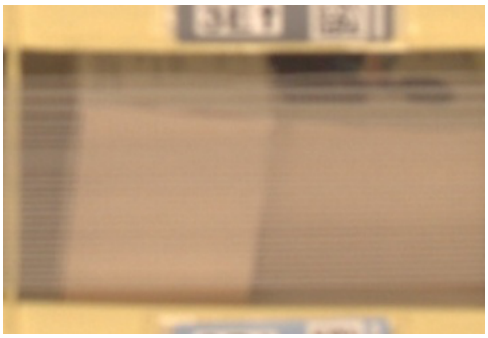
Fig 2: Example image from class 2

Fig 3: Example image from class 3

As shown , the images are all different sizes and the images themselves do not look very clear with regards to the number of objects within them. We believe this is a hard task that would be challenging for the human eye also.

The dataset was downloaded and split into training and test using a ratio of 80%: 20%.

Algorithms and Techniques

The python library PyTorch was used to conduct the modelling.

The classifier is a Convolutional Neural Network (CNN), this is state of the art for image recognition problems. This type of algorithm requires a vast amount of data for training which we fortunately have. A pre-trained network has been used and the final layer of the CNN was trained to produce the model. This is known as transfer learning. There are many implementations of pre-trained CNNs available in PyTorch, after some experimentation the model architecture chosen was ResNet50. This provided the highest accuracy for this dataset.

ResNet50 is a variant of ResNet model which has 48 Convolution layers along with 1 MaxPool and 1 Average Pool layer.[2]

Cross Entropy Loss was used to capture the loss of the model, this is suitable for multi class classification.

Benchmark model
A benchmark model was trained to view the increases (or decreases) in model accuracy from transfer learning. The benchmark model was a Resnet50 CNN without transfer learning, adding only 5 output neurons for predictions.

Final Model

The final model was a ResNet50 CNN with a feed forward fully connected network on top. The following architecture was chosen based on experimentation: Two connected layers and an output layer, the first with 475 neurons and the second with 325 neurons. The output layer has 5 neurons for predicting the class.

This model is known as the DL model.

Data Preprocessing

The preprocessing steps for the dataset included the following:

1. Resizing the images – this is required as we need uniform image sizes for training
2. Normalising the images – the neural network work better with floating point numbers with a much smaller range as opposed to 8 bit-integers between 0-255 used for pixel representation. This normalisation applies a mean of 0 and standard deviation of 1. This is also help increase speed to convergence of the model.
3. Random resize crop – this is a data augmentation which will help with generalisation of the model.

The training set has been shuffled however the test set was not shuffled.

[2] https://iq.opengenus.org/resnet50-architecture/

Hyperparamater Tuning

Hyperparameter tuning was applied to find the most optimal parameters based on the cross entropy loss. Tuning was conducted on learning rate, batch size and the number of epochs. The following is explained in the accompanying notebook:

The search range for each particular parameter has been chosen for the following reasons:

- *learning rate (lr): Controls how much to change the model in response to the loss when the model weights are updated. If too high the weights change too much and we increase error , if too low the model will take a long time to converge.*
- *batch size : We want the highest batch size that will allow for fast training but also without impacting the loss too greatly.*
- *epochs: We would like the best model without overfitting the model.*

Debugger and Profiler

These techniques important for model debugging and only available in Sagemaker allow us to determine how the model training is going by looking at cross entropy loss using debugger output and how computing performance is using profiler output.

Results

Hyperparameter Tuning Results

The results of the tuning are the following:
Learning rate : 0.072
Batch size: 64
Epochs: 3

These hyperparameters were uses for both the Benchmark and DL models.

Profiler Results

These reports are found in the repository accompanying this report. There were no major concerns on computing instances during the training of these models.

Benchmark model results

The following shows the debugger output from the benchmark model. It shows loss as the model trains. It can be seen that the blue line (the training loss) stays at a constant level as the training occurs. This tells me that the model is not improving after seeing all the training examples throughout the epochs. The validation loss behave the similarly but there is a cyclical pattern. This suggests there are validation examples which increase the loss that others do not. An inspection into the confusion matrix would help understand further.

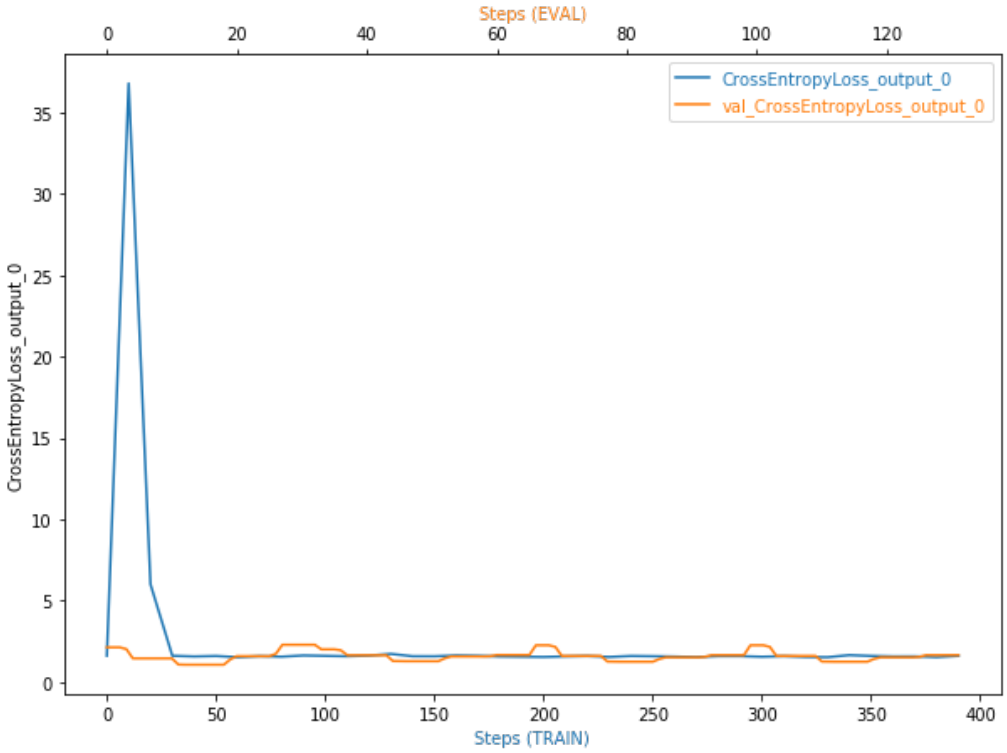

Fig 4: Benchmark debugging output

The confusion matrix tells us that the classifier seems to be predicting a high number of class 2 and class 4 versus all other classes.

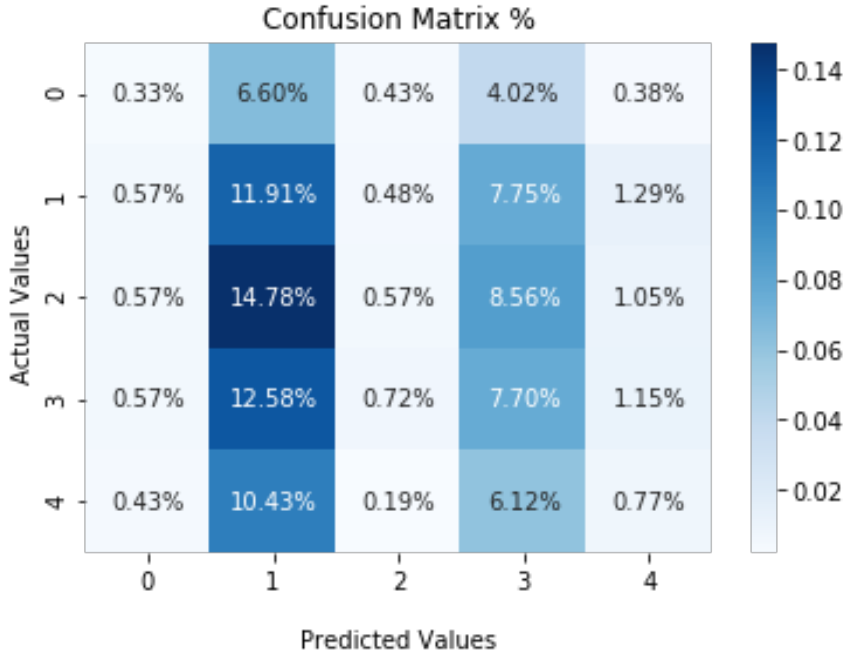

Fig 5: Benchmark confusion matrix

The metrics results table shows us class 2,3 and 4 have the highest precision and class 1 had the lowest precision, whereas class 2 and 4 have the highest recalls.

class	precision	recall	f1-score	support
1	0.135	0.028	0.047	246
2	0.212	0.541	0.304	460
3	0.240	0.022	0.041	534
4	0.225	0.339	0.271	475
5	0.165	0.043	0.068	375
accuracy			0.213	2090
macro avg	0.195	0.195	0.146	2090
weighted avg	0.205	0.213	0.157	2090

Table 1 : Benchmark metric results

Whilst it is good to see which classes the model has been the most confident for it is hard to ignore that the absolute results for this classification task are quite poor.

We will seek to understand if transfer learning has been able to increase performance of the models.

DL model results

The DL model debugging output shows the loss as the model trains. It's behaviour is similar to the benchmark model. The training loss does not decrease but stays steady and the validation error cyclical . It is worth looking into the metric to determine which class raises the loss.

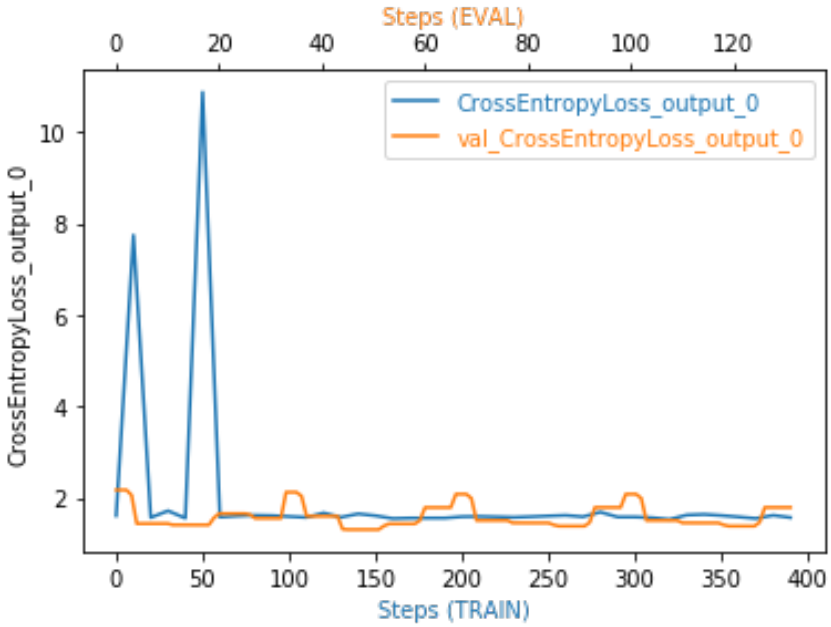

Fig 6: DL debugging output

The confusion matrix shows us the model is highly biased towards class 4 and that the model is barely predicting any other class.

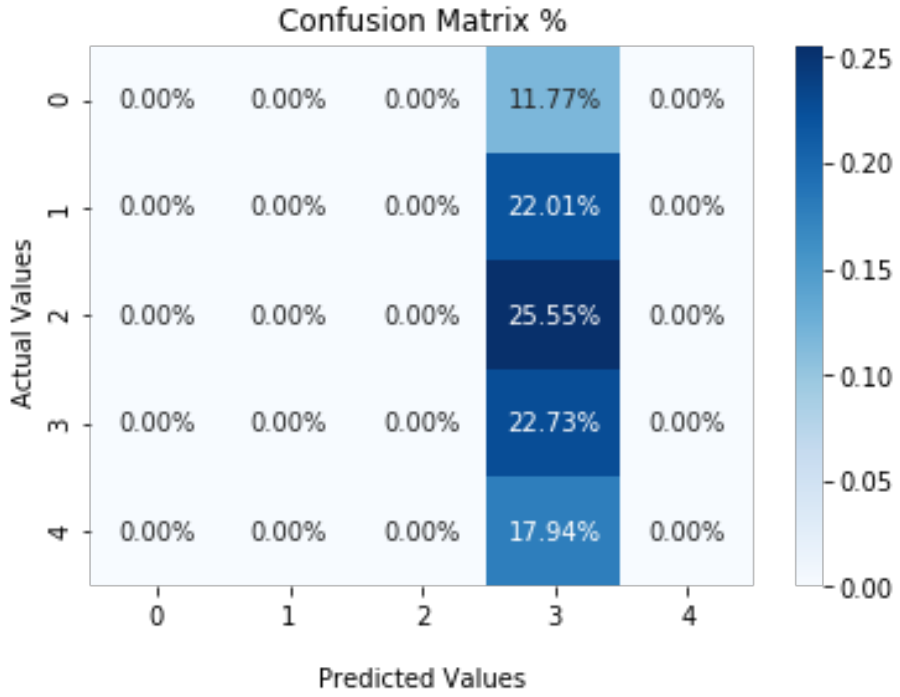

Fig 7 : DL confusion matrix

T
he metric results tell us that class 4 has perfect recall and precision is as best it can be given this.

class	precision	recall	f1-score	support
1	0.000	0.000	0.000	246
2	0.000	0.000	0.000	460
3	0.000	0.000	0.000	534
4	0.227	1.000	0.370	475
5	0.000	0.000	0.000	375
accuracy			0.227	2090
macro avg	0.045	0.200	0.074	2090
weighted avg	0.052	0.227	0.084	2090

Table 2: DL metric results

From these results it is easy to see that the DL model is a highly biased model and only predicts one class.

Reflection

This project has been surprising, it seems as though adding complexity with a feed forward fully connected network did not increase the modelling accuracy evenly, it just made the model concentrate on one class in particular. This suggests there is a lot of work needed with experimentation to find a suitable feed forward network that works for this kind of object counting problem.

Improvement

To improve the model , using proven techniques such as those used in crowd counting may be worth exploring[3]. As this project was testing the build of an end to end machine learning problem using AWS, the focus stayed on learning and using AWS . The tools used to build this project included S3, Sagemaker , Debugger , Profiler and Cloudwatch. Furtherwork, would entail deploying the model to and endpoint for predictions, using spot instances for training and multi-instance training.

[3] https://www.analyticsvidhya.com/blog/2021/06/crowd-counting-using-deep-learning/