CSE5DL Assignment Report

This report chronicles answers to questions raised while completing the CSE5DL assignment.

# Task 1

## Task 1a

### Data issues

## Several data issues have been examined across both training and validation dataset, including missing values, class imbalances, and label encoding. Neither dataset contains missing values, and the classes are well-balanced in both the training and validation datasets. The only issue identified was with the label encoding. To convert from one-hot encoding to integer labels, I used the argmax() function.

## Task 1b

### Why not use random\_split?

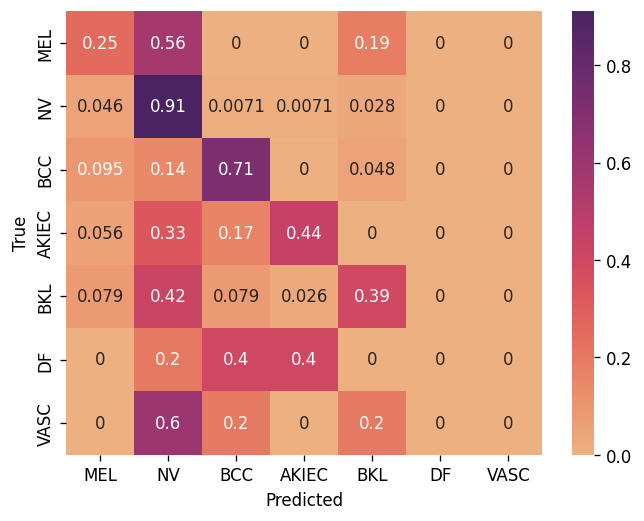
Random splitting is typically used to randomly divide a dataset into training, validation, and test sets. However, in our case, the dataset has already been pre-divided into training, validation, and testing sets. That’s why we do not need to use a random split.

## Task 1c

### Reduce epoch time.

### To significantly reduce epoch time for debugging purposes, two effective strategies include using a smaller subset of the data and reducing the batch size. A random fraction of the full dataset, such as 10%, can be used to create a smaller subset of the data. This approach reduces the time per epoch while still allowing us to work with real data, ensuring that the debugging process is both efficient and realistic. Besides, decreasing the batch size will lead to faster computations per epoch. Although this might slightly affect the model's convergence, it will enable quicker debugging cycles. Both methods allow the usage of real training data and real training code, making the debugging process more practical.

### Confusion matrix



The above confusion matrix shows the validation accuracy (0.75) and Unweighted Average Recall (UAR) (0.38) of the baseline model. From the confusion matrix, it is evident that there is a class imbalance. The baseline model failed to predict any instances of the DF and VASC classes. This indicates that during training, the model assigned less weight to the DF and VASC classes due to their small representation in the dataset.

## Task 1d

### Account for data issues.

## One of the data issues identified in question 1a was the underrepresentation of the DF and VASC classes compared to the other classes in the training dataset. This imbalance led to the model's inability to accurately predict these classes, resulting in poor validation Unweighted Average Recall (UAR). To address this issue, I implemented a weighted random sampler. This approach assigns appropriate weights to all the samples in the training dataset, ensuring that even classes with limited data are adequately represented during training. After implementing the weighted random sampler, the validation UAR improved significantly, and the trained model began to classify data across all classes more effectively. Although the model's UAR increased to 0.49, the validation accuracy slightly decreased from 0.75 to 0.71. Another way to address the class imbalance problem in the training dataset is to use torch.nn.BCEWithLogitsLoss and call the poss\_weight parameter. However, it is only appropriate for binary classification tasks. Since this task requires multiclass classification, the best option is to use a weighted random sampler to address class imbalance problems. The confusion matrix for the validation set is shown below:

A chart of different colored squares

Description automatically generated with medium confidence

## Task 1e

### Vertical Flips

 Random vertical flips randomly flip images vertically, which can distort the natural features of certain types of images. For instance, if we are training a neural network to detect different types of fruits and vegetables, using vertical flips would create upside-down images, which do not preserve the natural features of these objects. Training a model to detect upside-down fruits and vegetables would be a waste of computational resources and could negatively impact the model's performance.

However, in the context of cell classification, vertical flips are useful because cells can appear in any orientation. For our task of classifying skin lesion images, vertical flips are appropriate as they help augment the dataset in a way that reflects the natural variability in cell orientations. This ensures the model learns to recognize cells regardless of their orientation, improving its robustness and accuracy.

Effect of Augmentation

Data augmentation generates additional training data by applying various transformations, providing more samples for the model to learn from. While this can be beneficial, it does not always guarantee improved model performance. In some cases, data augmentation can create unrealistic samples, which can negatively impact test performance by training the model on distorted images.

For my training dataset, I applied random horizontal flips, random vertical flips, and random rotations as data augmentation techniques. Despite these efforts, the model's training accuracy dropped to 0.66, and the validation accuracy decreased to 0.62. However, the validation Unweighted Average Recall (UAR) improved from 0.49 to 0.59, indicating better class balance in predictions.

To compare the effects of data augmentation, I will present a screenshot of graphs from Weights and Biases below. The graphs present two models one from task 1d (non-augmented) and one from task 1e (augmented).

A group of graphs with text

Description automatically generated with medium confidence

[Challenge] 5 crop augmentation

I have skipped this task.

## Task 1f

### Experiments

* <See attached excel document> model\_training\_results.xlsx
* <See attached Weights and Bias Report> Weights\_and\_Biases\_Report

Weights and biases reports can be found in the ‘Weights and Biases Report’ folder. A total of 10 reports have been generated for 10 CNN models. Each report contains the graphs showing training accuracy, validation accuracy, training loss, validation loss, training UAR, and validation UAR along with a lot of other information.

* Write a discussion about the key findings from the experimental results.

A total of 10 different models were trained, including several ResNet models and custom models with various hyperparameters such as different learning rates, optimizers, and loss functions. The aim was to evaluate the performance of these models on a specific task, focusing on their training and validation accuracies.

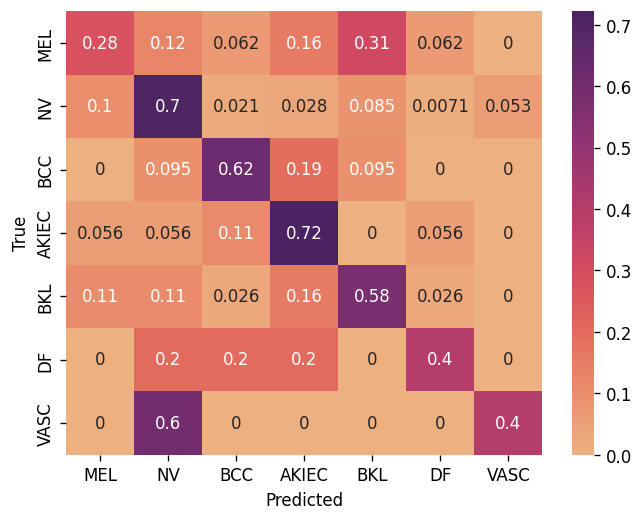
Four pre-trained ResNet models were trained: ResNet18 achieved a training accuracy of 0.46 and a validation accuracy of 0.59. ResNet34 showed similar training accuracy at 0.46 but had a slightly lower validation accuracy of 0.48. ResNet50 outperformed the other models, with both training and validation accuracies at 0.64. Lastly, ResNet152 reached a training accuracy of 0.63 and a validation accuracy of 0.60. All ResNet models were trained using a learning rate of 0.0001, the Adam optimizer, and the CrossEntropyLoss function. The results indicate that ResNet50 was the most effective among the pre-trained models.

The custom models were developed based on a baseline model but with increased filter sizes and two fully connected linear layers. Initially, a custom model with 8 layers using Conv2D, adaptive average pooling, and dropout with one fully connected layer was built, which resulted in poor performance (training accuracy of 0.49 and validation accuracy of 0.07). Subsequently, another custom model with 5 Conv2D layers, batch normalisation, max pooling, ReLU activation, increased filter sizes, and two fully connected layers was created. This configuration was tested with various learning rates. With a learning rate of 0.0001, this model, termed Custom Model2, achieved excellent results, with a training accuracy of 0.98 and a validation accuracy of 0.73. Increasing the learning rate to 0.001 (Custom Model3) led to a decrease in performance, with a training accuracy of 0.88 and a validation accuracy of 0.61. Further increasing the learning rate to 0.01 (Custom Model4) resulted in even poorer performance, with a training accuracy of 0.43 and a validation accuracy of 0.18. Using the SGD optimizer with a learning rate of 0.0001 (Custom Model5) also led to moderate performance, with training and validation accuracies of 0.59 and 0.56, respectively. The worst outcome was observed with Custom Model6, which used the NLLLoss function and achieved a training accuracy of 0.14 and a validation accuracy of 0.01. All models were trained using a random weighted sampler to ensure no class imbalance in the training data.

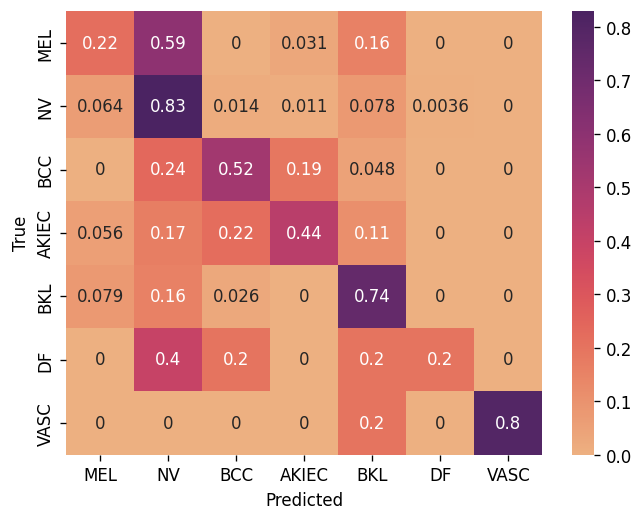
In conclusion, while ResNet50 outperformed other pre-trained models, Custom Model2 emerged as the best-performing model overall. All models were trained with a batch size of 64 for a total of 5 epochs.

To support the above explanations I will include two confusion matrices below, one from pretrained resnet50 and another from the best custom model.

**Resnet50 Confusion Matrix**:



**Custom Model2 Confusion Matrix:**

****

### [Challenge] Batch size

Assuming the full dataset is used in a single epoch, the number of updates per epoch is calculated as the total number of training samples divided by the batch size. If the batch size is halved, the number of updates per epoch will double. This can increase noise in the gradient estimates, which may help the model generalise better and escape local minima. The added noise acts as a regularizer, reducing overfitting, and is useful for small datasets or models prone to overfitting. However, this approach requires careful handling of the learning rate, as a smaller batch size can destabilise training. Reducing the learning rate can help maintain stability.

# Task 2

## Task 2a

### Data issues

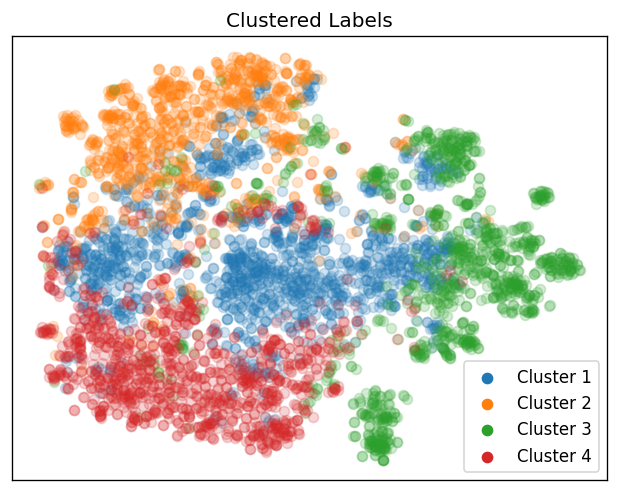
The given dataset did not have a header, so I added one using a pandas dataframe. As part of the data check, I verified that there are no null values in any of the columns. I also checked the unique values in the label column. The labels started from 1, but PyTorch expects labels to start from 0. To fix this, I subtracted 1 from each label. There were no issues with the texts. The dataset has two text columns including one with shorter texts, which are treated as headlines, and one with longer texts. The longer texts, along with the adjusted labels, are loaded into the dataset class for training.

## Task 2b

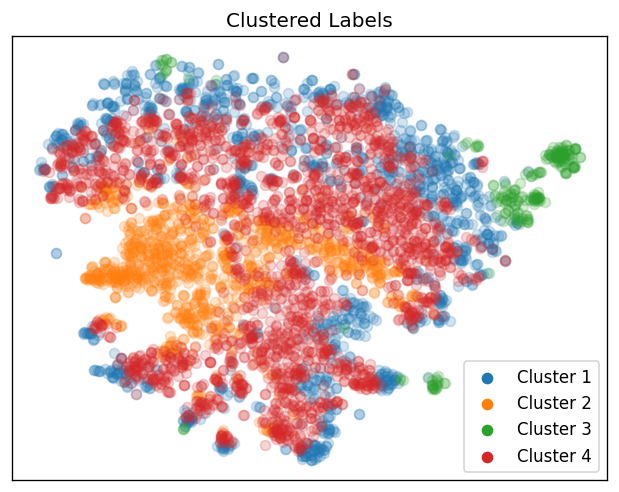
### Similar embeddings

By observing the sequence classification model and the token classification model, it is evident that the World and Sci/Tech classes have similar embeddings. This is indicated by the red (Sci/Tech) and blue (World) dots overlapping more with each other compared to other classes. This similarity likely arises because news articles in the World and Sci/Tech categories share many common words. Common vocabulary leads to similar representations in the embedding space.

**Sequence classification model:**



**Token classification model:**



## Task 2c

### Saved model weights

Despite creating only one `nn.Linear` layer, the saved model weights of a fine-tuned DistilBERT model are greater than 200MB. This is primarily because DistilBERT includes base model parameters such as 6 transformer layers, self-attention mechanisms, and embedding layers. Additionally, the pre-trained weights, which are fine-tuned during training, are stored along with the weights of the additional linear layer. This combination of extensive parameters and fine-tuned weights results in the saved model being over 200MB.

### What do the longs (int64) represent?

DistilBert models only accept input with a data type torch.int64. These longs (int64) represent tokenized text.

### [Challenge] Visualization of fine-tuned DistillBERT model.

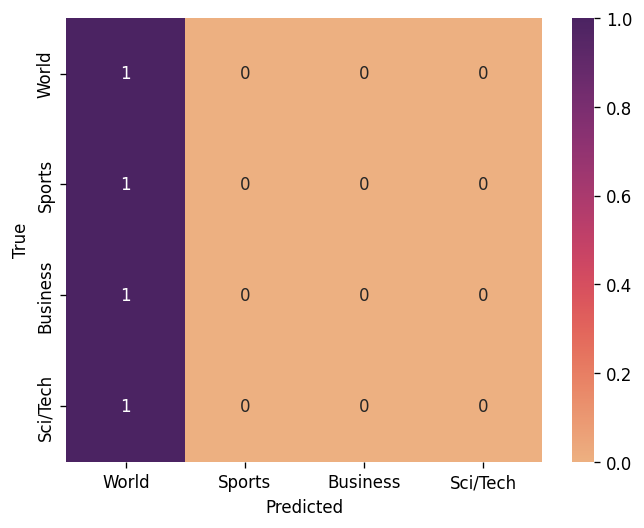
## The fine-tuned visualisation illustrates all the predicted labels overlapping with each other. Sci/Tech news embeddings dominating across all clusters.

## 

## Task 2d

### Class distributions and learning rate

After uncommenting the lines in train.py and training the DistilBert model again, I observed serious class imbalance. The class distribution suggests that class 2 is significantly underrepresented. Some batches mainly contain data of a single class, namely class1. Also, looking at the confusion matrix, it is easily observed that the model classified all news to one class (World), resulting in poor training accuracy of 0.26 and validation accuracy of 0.24, with a UAR of 0.25 for both. This issue can be addressed by fixing the learning rate. To determine whether to increase or decrease the learning rate, observe the training behaviour. If the loss is high or fluctuates significantly, the learning rate is likely too high. In this case, with CrossEntropyLoss, both training and validation losses were 1.38, which is high given that there are only 4 classes. This indicates that the learning rate was too high. The best approach is to decrease the learning rate and observe the results. Confusion matrix before fixing the learning rate:



### Relative performances before and after fixing learning rate

After decreasing the learning rate to 0.00001, the distilbert model was trained again. This time the training accuracy increased to 0.94 and validation accuracy increased to 0.90. UAR for both training and validation increased to 0.90.

The improvement was expected. Initially, the learning rate was too high, causing the model to struggle with convergence. The decreased learning rate 0.00001 allowed the model to update its weights more gradually and effectively, thus improving the overall performance. Confusion matrix after fixing the learning rate:

