**COMP309 Project Report**

The goal of this project is to use machine learning techniques to perform classification on a series of images to distinguish from a series of tomatoes, cherries and strawberries images which image belongs to which their respective class. The dataset used is comprised of 4500 images for training data (1500 of each class) and 1500 unseen test data. The approach is to identify outliers / noisy images and attempt to lessen their effect on the model’s predictive capabilities either using computer vision resizing techniques or removing samples (if necessary). To do this effectively means optimizing the convolutional neural network with the means to handle variations in backgrounds, image quality and lighting. To do this there will be emphasis on data preparation, identifying problematic feature in the dataset and apply further image filtering on blurry images to assist my model. Some challenges may include computational limitations (using multi layered neural networks with no GPU assistance) and sensory oversights (missing or irrelevant data due to human misinput).

**EDA and Preprocessing**

Upon inspecting the data less than 1.5% of the data for each class had images in need of resizing. After resizing and normalization images that did not match the 300x300 dimensions, there was an increase in problematic (noisy images), because these images are such a small representation of the data and contribute to noise, to avoid this the images were outright removed from the training data. Additionally, a 300x300 dimension normalisation is applied to the images upon loading them as a transformation after being loaded by Torch’s Image Folder data loading method. Upon investigating the images, there was a large number of blurry images particularly within the tomato and cherry classes. To identify blurry images, by applying the Laplacian operator derivative to each image and outputted the transformed image value’s variance, this can quantify the pixel intensity to distinguish blurry images. If the variance was lower than the specified threshold (50) then that image is listed as a ‘blurry’ image. With a threshold of 50, there were 87 blurry images of tomatoes, 8 blurry images of strawberries and 51 blurry images of cherries. In order to combat this issue, I decided to apply an image sharpening technique using computer vision’s filter2D function which takes a kernel (matrix) function and centres it to each pixel in the image then the kernel is applied. By my interpretation this process effectively sharpened the quality of previously very blurry images. The other prominent issue amongst the image data was the extremely dark or in less cases extremely bright images that tended to fade into the background. To approach this issue, a function was made to identify the mean of the pixels in the image to quantify the contrast of the image. All the values that fall under the threshold 20 (the underexposed or images that are too dark) are categorized as outliers, and the high threshold 230 pixels (overexposed or images that are too bright) are categorized as outliers. From this criterion I identified 23 outliers in the ‘tomato’ class, 28 outliers in the ‘strawberry’ class and 22 outliers in the ‘cherry’ class. Whilst these numbers are very low in the grander scope of the dataset, feature enhancement is being implemented in the case of extreme outliers in unseen data. The feature enhancement strategy chosen was to apply histogram equalization to enhance the contrast of problematic instances by redistributing the pixel values of the image to be more spread out. The method was effective for images that were closer to the threshold values, however, images with purely white or black backgrounds still fall in the outlier range post feature enhancement but the images seem to be ‘pop out’ more in comparison to prior to the transformation. As can be seen in the example images below.

A close up of a strawberry

Description automatically generatedA pair of red cherries

Description automatically generatedA tomato with a bug on it

Description automatically generated

The images fade into the background of the image far less after preprocessing. All the feature enhancement techniques worked effectively at correcting troublesome images. I hypothesize that because the features have become more distinct with image sharpening and contrast adjustments the basic MLP neural network (with only one hidden layer) will struggle to classify these instances into their respective classes. With just one layer the MLP may struggle to find the more complex defining features that will enable it to achieve great accuracy. However, these preprocessing transformations should enable the CNN with more specified architecture towards the image classification task to be far more effective in terms of accuracy. To prevent overfitting image augmentation was implemented to bring more diversity to the model, the techniques I chose were RandomHorizontalFlip, RandomRotation and ColorJitter. Because of the previously implemented functions the model won’t allow extreme changes in colour meaning the images should always fall within the specified threshold. These data augmentation techniques are applied transformations to random images enabling the model(s) to see the data with multiple perspectives and hopefully increase performance whilst more importantly preventing overfitting.

**Designing and Training CNN model**

**Validation for model tuning**

Prior to training the data I performed an 80:20 training, validation split on the training data to gauge the model’s performance in terms of generalisation. Ideally, with more computational resources it’d be appropriate to implement k-cross fold validation to optimally evaluate the model. Unfortunately, this approach exponentially increased computational resource expenditure which is not a goal for this model. Implementing the training, validation set split should suffice for the main purpose of this CNN model’s concern which is avoiding overfitting but not adding additional computational resources.

**Transfer Learning**

This CNN model was implemented on top of a prebuilt ResNet18 neural network, this was done to try and mitigate the vanishing gradient technique as much as possible, ResNet18 applies batch normalization which encourages good behaviour from the gradients. Additionally, and most importantly the use of ResNet18 comes with a pretrained weights parameter (pretrained on ImageNet dataset), which is extremely beneficial for this particular task as computing power is one of the model’s limitations (due to hardware). Using transfer learning enables the CNN model to utilise seeing a diverse range of images to better understand key features like edges, textures and more complex parts of the images. Some other considerations were efficient net, which is useful for its lower memory usage and lower computational times for image recognition and InceptionV3 for similar reasoning. Ultimately ResNet18 garnered the highest validation score of the three (85% validation accuracy prior to further implementations), so that was the pre-trained architecture that was used.

**Regularisation Techniques**

The next tuning strategy that was considered was the regularisation strategy, some of the techniques were considered were Dropout, where random neurons are dropped from the network each training batch. In order to save computational resources early stopping was also considered if the model’s convergence rate was sooner, the model can run for less amount of time and stop overfitting. Ultimately these techniques were decided against because they didn’t fit as naturally with the architecture (using transfer learning with a pre-trained model), there were some concerns about the interactions when running dropout when ResNet18 runs batch normalisation and the freezing layers approach to stopping overfitting has better synergy with transfer learning than stopping the model early. The weight decay was set to 1e-4 to also introduce a penalty to the loss function to reduce the magnitude of larger weights negatively influencing the model. By setting requires grad = false for all parameters we’re freezing the layers meaning the model can use the general features learned from training on ImageNet and focus training the model on the more specific to the task features that are found closer to the output layer in the CNN (in this case only the fully connected layer). Using this regularisation approach should help prevent overfitting but coincide with the architecture to enable the mode to generalise well to new unseen data.

**Loss functions**

The next aspect of tuning the model was deciding on a loss function that was appropriate for a multi-class classification project. Before training the model, I have a bias in favour of cross entropy loss as it is well optimized and incorporates SoftMax (ensures the z values all add up to a probability of one) into its output which is essential for a multi class classification problem. Another consideration for a loss function was Dice Loss, however it is not as suited for image multi class classification tasks and more so for image segmentation. The alternative optimizer chosen for this model was stochastic gradient descent with momentum of 0.9 and a learning rate of 0.001. Unsurprisingly, cross entropy loss (with label smoothing) yielded much better results on both the training and validation sets, this is unsurprising because cross-entropy loss is a function that has very good specificity towards multi-class classification tasks (particularly using PyTorch) measuring predicted output and the ground truth target classes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Epoch** | **Cross-Entropy Loss - Train Accuracy** | **Cross-Entropy Loss - Validation Accuracy** | **Dice Loss - Train Accuracy** | **Dice Loss - Validation Accuracy** |
| 1 | 72.99% | 88.63% | 68.21% | 85.47% |
| 2 | 86.23% | 90.08% | 78.56% | 87.21% |
| 3 | 88.35% | 90.64% | 81.12% | 88.03% |
| 4 | 88.80% | 91.53% | 82.77% | 88.93% |
| 5 | 88.96% | 91.53% | 83.45% | 89.15% |
| 6 | 89.02% | 90.19% | 83.97% | 88.82% |
| 7 | 89.99% | 92.42% | 85.10% | 89.47% |
| 8 | 90.11% | 91.97% | 85.87% | 89.32% |
| 9 | 90.91% | 91.75% | 86.50% | 89.05% |
| 10 | 91.22% | 92.08% | 86.88% | 89.23% |

In addition to showing that cross entropy loss had superior performance, these results give confidence in the model as the validation set and training set performances are comparable for both loss functions meaning there’s less chance the model is overfitting. The choice to use label smoothing stems from the earlier decision of using transfer learning, where generalisation and noise is a big concern, label smoothing ensures the outputs are not overly confident in their predictions (probability to predicted class is <1 and distributed to other classes).

**Mini Batch Size**

The final aspect of tuning the model I considered was the mini batch sizes, as training time and efficient computations is a priority, it seemed appropriate to investigate which mini batch results would yield the best performance. The figure above represents performance with mini batch equal to 32. When applying different minibatch sizes with the architectural designs of the model the same these were the results:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Epoch** | **Minibatch Size 16 - Cross-Entropy Train Acc** | **Minibatch Size 16 - Cross-Entropy Val Acc** | **Minibatch Size 64 - Cross-Entropy Train Acc** | **Minibatch Size 64 - Cross-Entropy Val Acc** | **Minibatch Size 128 - Cross-Entropy Train Acc** | **Minibatch Size 128 - Cross-Entropy Val Acc** |
| 1 | 70.35% | 87.22% | 73.67% | 88.45% | 75.23% | 87.65% |
| 2 | 85.02% | 89.56% | 85.89% | 89.97% | 86.10% | 88.90% |
| 3 | 87.45% | 90.15% | 87.75% | 90.60% | 88.12% | 90.00% |
| 4 | 88.55% | 91.12% | 88.65% | 91.34% | 88.70% | 90.97% |
| 5 | 88.70% | 91.22% | 88.98% | 91.49% | 89.05% | 91.15% |
| 6 | 88.93% | 90.76% | 89.55% | 90.78% | 89.30% | 90.42% |
| 7 | 89.45% | 92.00% | 90.35% | 91.95% | 90.00% | 91.87% |
| 8 | 89.72% | 91.45% | 90.57% | 91.80% | 90.20% | 91.52% |
| 9 | 90.35% | 91.22% | 91.00% | 91.65% | 90.80% | 91.25% |
| 10 | 90.87% | 91.55% | 91.23% | 91.92% | 91.05% | 91.38% |

Although the larger batch sizes had decent results, they didn’t yield a significant improvement over the mini batch size at 32, whereafter 10 epochs had a validation accuracy of 92%. Increasing batch size correlates to training taking longer which is not the goal of this project, whilst a minibatch size of 16 does show promising results the results of mini batch size of 32 are the best with only a marginal increase in sizing.

**Performance evaluation (Comparison to MLP)**

The trained MLP model was kept extremely basic the purpose of which to provide a baseline for the CNN network, this was reflected in the performance (according to accuracy on both training and validation sets). The MLP neural network only consisted of one hidden layer using an ReLU activation function (also used cross entropy loss as loss function). Both the CNN and MLP neural networks used the same optimizer Adam to hold for comparisons. These were the respective results of both the neural networks:

MLP: A screenshot of a graph

Description automatically generated

CNN: A screenshot of a graph

Description automatically generated

The MLP had terrible accuracy even in later epochs, this is to be expected, because this model is not optimized like the CNN model and was kept extremely simplistic. In addition, MLP neural networks are less able to pick up on spatial features like distance, direction and complex shapes compared to a CNN. This is because of how the neural network is built, MLPs flatten images into one dimensional vector, meaning neighbouring pixels lose positioning to each other. In addition, changing the structure of the image makes it more difficult for the model to identify important patterns in image classification like edges or shapes. This resulted in a poor final validation accuracy of 46.15% accuracy. The CNN model achieved significantly better results, factors that influence this are the optimization such as layer freezing, use of pre-trained neural networks to leverage features and the use of layer smoothing with the loss function. In addition, mechanisms of a CNN model enable it to be far more useful for image classification, rather than being fully connected neurons, each neuron in the convolutional layer acts as a sliding window (filter) focusing on local features. Using the approach of identifying the local features, then applying the non-linearity and then down sampling is a far more effective structure for image classification, as the network goes deeper it’s able to identify more complex features. Because this convolutional neural network has these mechanisms and additional optimization/ tuning techniques applied it is unsurprising it yielded a much better performance than the MLP with 92.08% accuracy on the validation set.

**Conclusion**

Within this project, we created a successful convolutional neural network designed for multi-class classification with the goal of classifying strawberries, tomatoes and cherries. Through investigation via EDA, we identified some preprocessing techniques to prepare the dataset such as image resizing, contrast adjustments and image sharpening. One of the key objectives of this project was to prevent overfitting whilst also minimizing computational output, using transfer learning and regularisation techniques such as weight decay and batch normalisation the model achieved incredible results on the validation set (92% accuracy). For future endeavours into image classification, it’d be beneficial to investigate different optimization techniques and consider other activation functions. Despite having good performance there is still room for improvement with more computational resources to introduce concepts such as ensemble learning but overall this project achieved satisfactory results.