Lab Assignment - 2

Rohan Eswara

1 Questions

1. **Influence of Regularization [15 points]:**

   (a) Design and conduct your experiment (Code)

      i. Load a real dataset not covered in the course that is designed for the multi-class classification problem; for example, from sklearn.datasets, Kaggle, your own data. If you are seeking assistance for how to find a dataset, one good option is to revisit the TA's programming tutorials. You can pre-process the dataset if you wish.

      ii. If the dataset you choose does not already come with a train/validation/test split, then divide the dataset into a 70/15/15 train/validation/test split.

      iii. Train all combinations of at least two optimization approaches with three regularization approaches, resulting in at least 6 trained models (i.e., 2 x 3 = 6 models). Optimization approaches can include those covered in the course, such as vanilla, step decay, exponential decay, AdaGrad, RM- Sprop, and Adam. Regularization approaches can include those covered in the course, such as parameter norm penalties, early stopping, data augmentation, dropout, and batch normalization. Set all other hyperparameters constant when training; for example, number of iterations for training and batch size.

      iv. As a baseline for comparison, also train two models without regularization. When training these models, everything should be identical to the previous step (optimization approaches, neural network architectures and hyperparameters) except that no regularization technique is used.

      v. Evaluate each of the 8 trained models on the validation set with respect to how long it took to train and the accuracy metric.

      vi. Train a new model on all of the training and validation data using the top-performing neural network architecture and training protocol from all models tested on the validation set. Then, evaluate the resulting model on the test set using the accuracy metric and a confusion matrix.

      vii. Note: All methods and results from this coding section must be presented separately in your report, as described below, to receive full credit.

   (b) Report your methods, results, and analysis (Report) ;

      i. Describe the methods you used for your experiment such that the reader could reproduce your experiments. This should include a discussion of the 1

dataset (such as source? number of examples?), neural network architectures, and training process (including all relevant hyperparameter values) used to train all the models. Please note that the description should include default settings inherited from the programming language.

ii. Report for each of the initial eight models how long it took to train and the corresponding accuracy scores.

iii. ndicate which model configuration led to the top-performing model on the validation set and report the performance of the final model that was evaluated on the test split using both the accuracy score and confusion matrix.

iv. Discuss your analysis of the results. Your discussion should consist of 2-4 paragraphs. This can include general trends that emerge from your results, in which case please first report the trend and then offer insights/speculations into why you think the trend/results may occur (regardless of whether you deem the results good or bad). Possible trends to consider include: What is observed when comparing the use of regularization techniques to not using any regularization during training? Did certain regularization techniques or optimization schemes lead to consistently better results? What do you see as the trade-offs between training time and different choices for the regularization techniques? You also can discuss your analysis of the final top-performing model configuration. For example, how does it compare when tested on the validation set versus the test set? What insights are gained by looking at the different evaluation metrics for the final model?

(c) **CNNs - Pre-Trained vs Fine-Tuned Models [10 points]:**

i. Design and conduct your experiment (Code)

A. Load a real dataset not covered in the coding tutorials that is designed for an image classification problem. If you are seeking assistance for how to find a dataset, one good option is to revisit the TA's programming tutorials. You can pre-process the dataset if you wish.

B. If the dataset you choose does not already come with a train/test split, then divide the dataset into a 70/30 train/test split.

C. Load a pre-trained image classification model not covered in the coding tu- torials to support prediction after fine-tuning. For fine-tuning, modify the model's final fully-connected layer(s) to match that for your dataset and then fine-tune the model parameters for your dataset. You can decide which hy- perparameters to use when training, such as the optimization scheme. During training, analyze its learning process by creating a plot with two curves for both the training and validation splits. This plot should show what happens to the model's accuracy as the number of training epochs increases when training the neural network. Choose as the final fine-tuned model the one you believe performs the best based on the learning curves, which means that during training you will need to

ensure you are storing "checkpoints" of the model parameters that you can later retrieve.

D. Note: All methods and results from this coding section must be presented separately in your report, as described below, to receive full credit.

ii. Report your methods, results, and analysis (Report)

A. Describe the methods you used for your experiment such that the reader could reproduce your experiments. This should include a discussion of the dataset (such as source? number of examples?), neural network architectures, and training process (including all relevant hyperparameter values) used to train all the models. Please note that the description should include default settings inherited from the programming language.

B. Report the learning curves for the fine-tuned model as well as the accuracy score on the test set for the final, chosen fine-tuned model.

C. Discuss your analysis of the results. Your discussion should consist of 1-3 paragraphs. Each paragraph start with one or more sentences describing the observed results. Subsequent sentences in each paragraph should then offer insights/speculations into why you think the results may occur (regardless of whether you deem the results good or bad). Possible trends to consider include: What is observed when comparing the use of a pre-trained versus fine-tuned model? What, if any, insights are gained by looking at the learning curves of the fine-tuned model (such as overfitting vs underfitting)? What, if anything, can you infer about what visual concepts the models learned?

2 Influence of Regularization

2.1 Methods

In the first part of the assignment we were required to use a real multiclass classification dataset that has not been utilised in class to test the influence of both regularization techniques and optimization algorithms on the performance of Neural Networks. All other hyperparameters from the ones mentioned above are kept constant to make sure that we are highlighting the influence of these parameters on the performance of Neural Networks.

2.1.1 Dataset

In this section of the assignment, a dry bean multiclass classification dataset is used. This dataset was obtained from Kaggle. In the research that led to the compilation of this dataset, seven distinct types of dry beans were meticulously selected, considering various attributes such as form, shape, type, and structure based on market considerations. The primary objective of the research was to create a robust classification system capable of distinguishing between the seven registered varieties of dry beans that share similar visual

characteristics. To achieve this, a sophisticated computer vision system was developed, leveraging high-resolution imaging technology. The system captured images of a total of 13,611 grains, representing the seven different registered varieties of dry beans. These bean images underwent a comprehensive processing pipeline, including segmentation and feature extraction stages. Through this process, a set of 16 features was obtained, comprising 12 dimensions and 4 shape forms. All of the 16 features of this dataset are numerical features and are mainly used to describe the different dimensions of a dry bean.

This dataset's sixteen features are described below :

1. **Area:** Represents the surface area of a defined bean region, indicating the number of pixels within its boundaries.
2. **Perimeter:** Denotes the circumference of a bean, calculated as the length of its border.
3. **MajorAxisLength:** Refers to the distance between the endpoints of the longest line that can be drawn within a bean region.
4. **MinorAxisLength:** Represents the length of the longest line that can be drawn from the bean, oriented perpendicular to its main axis.
5. **AspectRation:** Describes the relationship between the length (L) and width (l) of a bean region.
6. **Eccentricity:** Indicates the eccentricity of the ellipse with the same moments as the bean region.
7. **ConvexArea:** Signifies the number of pixels within the smallest convex polygon that can encompass the area of a bean seed.
8. **EquivDiameter:** Represents the diameter of a circle with an equivalent area to that of a bean seed.
9. **Extent:** Denotes the ratio of pixels within the bounding box to the total bean area.
10. **Solidity:** Also referred to as convexity, it represents the ratio of pixels in the convex shell to those within the beans.
11. **Roundness:** Calculated using the formula $\frac{4\pi A}{P^2}$, providing a measure of the circularity of a bean.
12. **Compactness:** Measures the roundness of an object using the ratio $\frac{Ed}{L}$, where E is the circumference and d is the diameter.
13. **Shape Factor 1, 2, 3, 4:** Unfortunately, information regarding these factors is currently unavailable.

This dataset's target variable is a categorical one that represents the seven classes of dry beans that are present in the dataset. The seven classes are listed below :

1. There are 3546 instances of the **Dermason** dry bean class,
2. There are 2636 instances of the **Sira** dry bean class,

3. There are 2027 instances of the **Seker** dry bean class,
4. There are 1928 instances of the **Horoz** dry bean class,
5. There are 1630 instances of the **Cali** dry bean class,
6. There are 1322 instances of the **Barbunya** dry bean class, and
7. There are 522 instances of the **Bombay** dry bean class.

Suitable for a supervised multi-class classification machine learning task, this datatset can be used to train a model to perform the task of dry bean classification, where the different classes are mentioned above.

2.1.2 Data Preprocessing

Prior to feeding the data into the neural network, a series of preprocessing steps were applied to the dataset to ensure a decently optimized performance. Since all of the features of this dataset are numerical variables, they are subjected to normalization using the Min-Max scaling technique. This process, facilitated by the MinMaxScaler from the scikit-learn library, scales the values of each feature to a range between 0 and 1. Normalizing the features helps mitigate the impact of varying scales and ensures a consistent input for the neural network.

Additionally, the target labels were encoded using the Label Encoder from scikit-learn. This transformation converts categorical labels into numerical format, making them suitable for ML and DL models. Subsequently, the categorical labels were further one-hot encoded using functions from Keras utilities, creating a binary matrix representation.

To evaluate the model's performance, the dataset was split into training, testing, and validation sets using the TrainTestSplit function. Initially, the dataset was divided into training (70 percent) and testing (30 percent) sets. Further, to establish a validation set for model tuning, the testing set was again split into validation (15 percent of the entire dataset) and test (15 percent of the entire dataset) subsets. The random seed (42) was set to ensure reproducibility in the split, allowing for consistent evaluation metrics during model development. The processed data (training, testing, and validation sets), consisting of scaled features and one-hot encoded labels, is now ready for training, evaluating, and fine-tuning the Neural Network.

2.1.3 Model Architectures :

In this section, the model architectures of all the eight different configurations obtained by combining the two different optimization approaches and the three different regularization techniques, and the two baseline models (without any regularization) are described. The base architecture remains consistent across all eight models, serving as a common framework upon which diverse optimization algorithms and regularization techniques are subsequently implemented. The base architecture includes an input dense layer, consisting of 256 units and employing the Rectified Linear Unit (ReLU) activation function, serves as the initial receptive field for the features of the training data. Subsequently, a hidden layer with 128

units, also activated by ReLU, facilitates the extraction of deeper representations from the learned features. After the first hidden layer, another hidden layer with 64 units activated by ReLU is added to extract higher level features. The output layer, composed of 7 units and employing the softmax activation function, generates probabilistic predictions across the seven different classes. All of the eight models use the categorical crossentropy loss function to optimize their parameters. Each model takes input data in the form of batches of size 32. The two different optimizers used are RMSProp and Adam, and the three regularization techniques used are Dropout, Batch Normalization, and Early Stopping. The learning rate used for both the optimizers in this section of the assignment is 0.001, which is the default value for both the optimizers Adam and RMSProp.

The following paragraphs describe the model architecture of each of the eight models :

1. **Model 1 - No regularization and Adam optimizer :** Model 1 adopts the base architecture as the foundational framework, consistent across all eight configurations. This architecture starts with an input layer comprising 256 units and employing the Rectified Linear Unit (ReLU) activation function, serving as the initial receptive field for the features within the training data. Following this, a hidden layer with 128 units, also activated by ReLU, enables the extraction of deeper representations from the acquired features. Following this, a hidden layer with 64 units, also activated by ReLU, enables the extraction of Higher Level Features from the input that is passed to this layer. The output layer, consisting of 7 units and utilizing the softmax activation function, generates probabilistic predictions across seven distinct classes. Model 1 employs the Adam optimizer and the categorical crossentropy loss function for parameter optimization. The model is trained with a batch size of 32. Model 1 does not employ any regularization techniques.

2. **Model 2 - No regularization and RMSProp optimizer :** Model 2 is exactly the same as Model 1, except it uses the RMSProp optimzer instead of the Adam optimizer.

3. **Model 3 - Dropout regularization and Adam optimizer :** Model 3 follows the base architecture outlined in the previously. In this configuration, Dropout regularization is employed to enhance the model's generalization capability. The base architecture begins with an input layer consisting of 256 units, each activated by the Rectified Linear Unit (ReLU) function, establishing an initial receptive field for the training data features. Subsequently, a Dropout layer with a rate of 0.1 is introduced to randomly drop a fraction of the units during training. A hidden layer with 128 units, also activated by ReLU, contributes to the extraction of deeper representations from the learned features. Another Dropout layer with a rate of 0.1 is applied to further encourage robustness. This is followed by another hidden layer with 64 units that employs the RELU activation function. The final layer, composed of 7 units and utilizing the softmax activation function, generates probabilistic predictions across the seven distinct classes. Model 3 is compiled using the Adam optimizer and the categorical crossentropy loss function to optimize parameters. The model is trained with batches of size 32.

4. **Model 4 - Dropout regularization and RMSProp optimizer :** Model 4 is

exactly the same as Model 3, except it uses the RMSProp optimzer instead of the Adam optimizer.

5. **Model 5 - Batch Normalization regularization and Adam optimizer :** Model 5 adopts the shared base architecture. The base architecture comprises an input layer with 256 units utilizing the Rectified Linear Unit (ReLU) activation function, initiating the model's interaction with training data features. Integral to Model 5 is the incorporation of Batch Normalization layers, strategically positioned after each densely connected layer. These Batch Normalization layers aim to enhance convergence and stability by normalizing the input to each layer during training. Following the input layer, a hidden dense layer with 128 units, activated by ReLU, facilitates the extraction of intricate representations from the learned features. This hidden layer is followed by another hidden layer with another hidden layer with 64 units, which is also activated by the RELU activation function. The output layer, consisting of 7 units and employing the softmax activation function, generates probabilistic predictions across seven distinct classes. Model 5 utilizes the Adam optimizer and categorical crossentropy loss function for parameter optimization, with input data processed in batches of size 32 during training.
6. **Model 6 - Batch Normalization regularization and RMSProp optimizer :** Model 5 is exactly the same as Model 5, except that it uses the RMSProp optimization algorithm instead of the Adam Optimization algorithm.
7. **Model 7 - Early stopping regularization and Adam optimizer :** Model 7 adheres to the shared base architecture, providing a consistent foundation for all configurations. The base architecture consists of an input layer with 256 units utilizing the Rectified Linear Unit (ReLU) activation function, serving as the initial receptive field for training data features. Subsequently, a hidden layer comprising 128 units, also activated by ReLU, facilitates the extraction of intricate representations from learned features. This is followed by another hidden layer that has 64 units and utilises the RELU activation. The output layer, consisting of 7 units and employing the sigmoid activation function, generates probabilistic predictions across seven distinct classes. For Model 7, the optimization approach adopts the Adam optimizer, and the categorical crossentropy loss function is employed to optimize model parameters. Notably, the training data is processed in batches of size 32. The inclusion of early stopping, with a patience of 10 epochs while monitoring the validation loss, enhances the model's training efficiency and prevents overfitting during the learning process.
8. **Model 8 - Early stopping regularization and RMSProp optimizer :** Model 8 is exactly the same as Model 7, except that the RMSProp algorithm is used to optimize the model's parameters rather than the Adam algorithm.

2.1.4 Methodology :

Each of the eight models were trained and tested on the training set and the validation set. The training process was carried out for 150 epochs for each model, and the inputs to each model were divided into batches of size 32. The time taken to train and validate each of the

eight models were also recorded. The time taken to train and validate each model is given in the results section below. The model out of the 8 different configurations that performed the best on both the training and validation set was selected for the testing phase.

In the testing phase, the model that performed the best (in our case Model 3 - Dropout regularization and Adam optimizer) was selected and was evaluated on the test dataset that was generated during the data pre-processing stage. The reason for choosing Model 3 as the best performing model is explained in the results section.

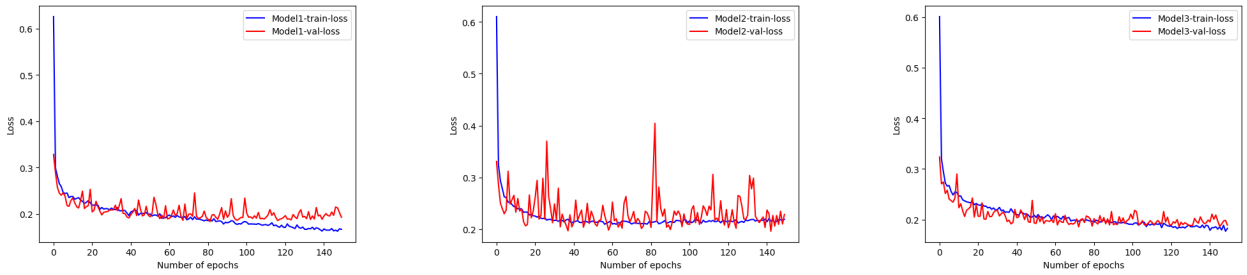
2.2 Results

This section describes the training and validation metrics obtained for each of the eight models, the time taken for the training and validation phase for each of the eight models, and the evaluation or testing metrics of the best performing model.

The training and validation metrics obtained for each of the eight different configurations are training loss, validation loss, training accuracy, the validation accuracy, and the time taken to complete the training and validation process. The training and the validation metrics for each of the eight models are shown in the table 1.

Now let us go through the training and validation metrics each of the eight models that were trained, and how we chose our best performing model.

1. **Model 1 - No regularization and Adam optimizer :** As seen in figure 1a, the model starts to overfit slightly as the number of epochs gradually increase, especially as the number of epochs approaches 150. As the number of epochs approaches the maximum (150 in our case) the validation loss, although it oscillates, it remains constantly within a range, however the training loss is still reducing, this shows that the model 1 has slightly overfit on the available training data. Therefore we can say that this is not the most optimised model. The oscillation of the loss curves can be caused by high learning rate, small batch size or increased model complexity.
2. **Model 2 - No regularization and RMSProp optimizer :** As seen in figure 1b, the training loss curve and the validation loss curve for model 2 seem to oscillate a lot more than model 1. Although, the loss curves indicate that the model is not overfitting, the validation loss after the last epoch is not the lowest validation loss recorded amongst all the eight models, therefore we can say that this is not the most optimised model. The oscillation of the loss curves can be caused by high learning rate, small batch size or increased model complexity.
3. **Model 3 - Dropout regularization and Adam optimizer :** As seen in figure 1c, the loss curves indicate that the model is not overfitting or underfitting. Model 3 also has the lowest validation loss, and the highest validation accuracy recorded amongst the eight different configurations as shown in table 1. Therefore, this was selected as the best performing model on the training and validation dataset.
4. **Model 4 - Dropout regularization and RMSProp optimizer :**As seen in figure 2a, although the loss curves indicate that the model is not overfitting or underfitting,



(a) Loss curve for Model 1 (b) Loss curve for Model 2 (c) Loss curve for Model 3

Figure 1: Overall caption for the figure.

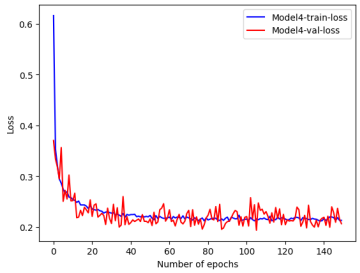
the validation loss of model 4 was not the lowest amongst the eight different NNs, therefore it was not the best performing model.

5. **Model 5 - Batch Normalization regularization and Adam optimizer :** As seen in figure 2b, the model starts to overfit slightly as the number of epochs gradually increase, especially as the number of epochs approaches 150. As the number of epochs approaches the maximum (150 in our case) the validation loss, although it oscillates, it remains constantly within a range, however the training loss is still reducing, this shows that the model 5 has slightly overfit on the available training data. Therefore we can say that this is not the best performing model. The oscillation of the loss curves can be caused by high learning rate, small batch size or increased model complexity.

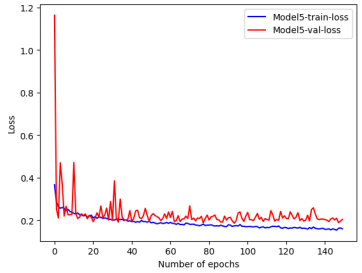
6. **Model 6 - Batch Normalization regularization and RMSProp optimizer :** As seen in figure 2c, the model starts to overfit slightly as the number of epochs gradually increase, especially as the number of epochs approaches 150. It is also seen that the validation loss curve of this model oscillates more than model 5. As the number of epochs approaches the maximum (150 in our case) the validation loss, although it oscillates, it remains constantly within a range, however the training loss is still reducing, this shows that the model 6 has slightly overfit on the available training data. Therefore we can say that this is not the best performing model. The oscillation of the loss curves can be caused by high learning rate, small batch size or increased model complexity.

7. **Model 7 - Early stopping regularization and Adam optimizer :** As seen in figure 2c, the model stops training after epoch 71 right before it starts to overfit. However, the validation loss and validation accuracy for this model are not the best observed amongst the eight different NNs that were trained. Therefore, we can say that model 7 is not the best performing model

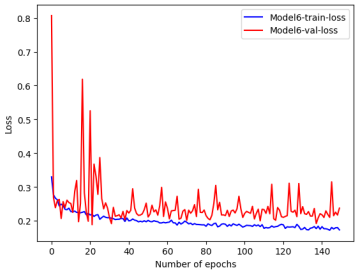
8. **Model 8 - Early stopping regularization and RMSProp optimizer :** As seen in figure 3b, the model stops training after epoch 42 right before it starts to overfit. However, the validation loss and validation accuracy for this model are not the best observed amongst the eight different NNs that were trained. Therefore, we can say that model 8 is not the best performing model.



(a) Loss curve for Model 4

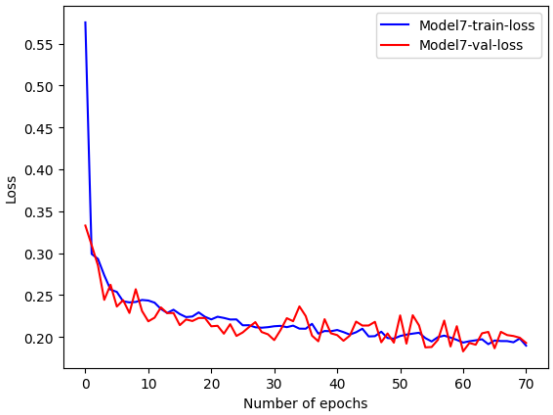


(b) Loss curve for Model 5

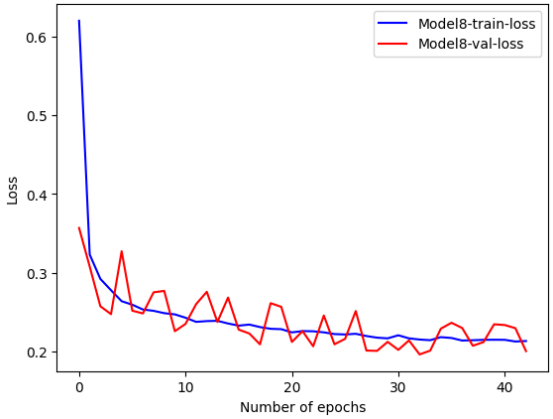


(c) Loss curve for Model 6

Figure 2: Overall caption for the figure.



(a) Loss curve for Model 7



(b) Loss curve for Model 8

Figure 3: Overall caption for the figure.

Model no	Train Loss	Train Accuracy	Val Loss	Val accuracy	time taken (s)
Model 1	0.1668	0.9331	0.1928	0.9329	232
Model 2	0.2191	0.2191	0.2285	0.9285	184
Model 3	0.1824	0.9278	0.1890	0.9339	263
Model 4	0.2130	0.9242	0.2065	0.9305	221
Model 5	0.1592	0.9384	0.2034	0.9300	324
Model 6	0.1730	0.9366	0.2373	0.9236	263
Model 7	0.1897	0.9265	0.1930	0.9310	113
Model 8	0.2131	0.9211	0.2003	0.9270	54

Table 1: Training and Validation metrics of all the eight different configurations

As seen previously, model 3 was chosen as the best performing model. This model was then evaluated against the test dataset. The tables 2 and 3 represents the classification report and the confusion matrix for model 3. The model 3 was able to obtain a test accuracy of **0.931**.

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.92	0.89	0.90	197
1	1.00	1.00	1.00	83
2	0.94	0.92	0.93	240
3	0.92	0.94	0.93	534
4	0.95	0.96	0.96	287
5	0.98	0.94	0.96	296
6	0.88	0.90	0.89	405
Accuracy			0.93	2042
Macro Avg	0.94	0.94	0.94	2042
Weighted Avg	0.93	0.93	0.93	2042

Table 2: Classification Report for model 3

	Class 0	**Class 1**	**Class 2**	**Class 3**	**Class 4**	**Class 5**	**Class 6**
Class 0	175	0	12	0	0	0	10
Class 1	0	83	0	0	0	0	0
Class 2	12	0	221	0	5	0	2
Class 3	0	0	0	502	3	5	24
Class 4	1	0	2	3	276	0	5
Class 5	2	0	0	9	0	278	7
Class 6	0	0	1	30	6	2	366

Table 3: Confusion Matrix for model 3, Class 0 (BARBUNYA), Class 1 (BOMBAY), Class 3 (Cali), Class 4 (Dermason), Class 5 (Horoz), Class 6 (Seker), Class 7 (Sira). The columns indicate the predicted classes, and the rows indicate the actual classes.

2.3 Analysis

In the first section of the assignment, an extensive experiment was conducted to observe the effects of regularization techniques on the training and validation performance of the model. On analyzing the results the following trends were observed.

Firstly, it was seen that when the model that used Adam optimizer without any regularization was trained and validated, this model started to overfit slightly. However, when the models that utilized the Adam optimizer with a regularization technique were trained and validated, it was seen that the generalization gap decreased, thereby indicating that the models were less overfit when compared to the model without any regularization technique. This shows that

by incorporating these regularization techniques, we are essentially introducing mechanisms to prevent the model from fitting the training data too closely and encourage more robust learning patterns. As a result, the model becomes better at generalizing to new, unseen data, leading to a decrease in the generalization gap.

However, when we compared the model that utilized the RMSprop optimizer without regularization techniques with the models that used the same optimizers with regularization techniques, the results obtained did not show any kinds of trends. In fact, Model 2 vs Model 6 shows that with addition the batch normalization regularization technique to the model, the model started to overfit slightly.

Secondly, it was observed that the addition of Dropout regularization technique to models when used with both optimizers showed an improvement in the validation loss metric and the validation accuracy metric (Model 1 vs Model 3 and Model 2 vs Model 4). This could be due to the nature of the Dropout regularization technique, by randomly dropping out neurons, the network is encouraged to learn more robust and general features that are useful across different examples. Dropout can also be viewed as training an ensemble of multiple subnetworks. Each dropout mask during training corresponds to a different subnetwork. During inference, the predictions are averaged or combined, providing a more robust prediction.

Thirdly, it was observed that in case of models that used the early stopping mechanism the time taken to train the data was significantly lesser than the models that did not use early stopping. Although early stopping did prevent models from overfitting, it could not achieve the best validation metrics and training metrics when compared to other models (especially models that used dropout). This could be because, early stopping prioritizes the model's ability to generalize to unseen data (validation set) over its performance on the training set. Therefore, the training process may stop before the model achieves its optimal performance on the training set. The "patience" hyperparameter in early stopping defines the number of consecutive epochs without improvement required before terminating the training; if set too low, it may prematurely halt the training process before the model reaches optimal weights.

In addition, it was also observed that the models that used batch normalization technique when compared with the models that did not use any regularization techniques took more time to train, yet could not achieve the best possible training performance. This could be due to the additional computations involved in normalizing activations at each layer might contribute to increased training time. Batch normalization may change the optimal learning rate for training. If the learning rate is not adjusted accordingly, it can affect convergence speed and final performance.

Additionally, it was observed that the model that performed the best (Model 3 - Dropout regularization with Adam optimizer) performed equally well on the testing and validation dataset. The testing accuracy was 0.931, whereas the validation accuracy that was observed was 0.9339.

It was also seen that class 1 (Bombay) had perfect precision and recall, suggesting the model performs exceptionally well on this class, and class 5 (Seker) has high precision and recall,

indicating accurate and comprehensive predictions. The confusion matrix provides a more detailed view of the model's performance on individual classes. Classes 0 (Barbunya), 2 (Cali), and 4 (Horoz) show strong diagonal elements, suggesting good performance. Some confusion is observed in Class 3 (Dermason) , where false positives (24) and false negatives (8) are relatively higher. Class 3 (Dermason) might benefit from further investigation. Strategies such as data augmentation, fine-tuning the model, or adjusting class weights could help improve performance on this class.

3 CNNs - Pre-Trained vs Fine-Tuned Models

3.1 Methods

In the second part of the assignment, we were required to utilise an image classification dataset to fine-tune a pre-trained model for a specific task. After training the fine-tuned model, we are required to evaluate and analyse the trends observed by testing this fine-tuned model. The task chosen for this section of the assignment was a garbage classification task, where the images of garbage should be classified into one of the six possible classes (More about these classes in the dataset sub-section).

3.1.1 Dataset

In the second section of the assignment, the garbage classification image dataset obtained from kaggle was used. This dataset is a multi-class classification dataset that contains data samples for six different types of garbage including cardboard, glass, metal, paper, plastic and trash. There are totally 2527 total data points in this dataset, where 393 belong to the cardboard class, 491 to the glass class, 400 to the metal class, 584 to the paper class, 472 to the plastic class, and 127 belonging to the trash class.

3.1.2 Data prerpocessing and loading

Since the model was trained on colab, first, the garbage classification dataset was first uploaded to google drive. Then each image in the dataset was first loaded into a variable, transformed into an image array using keras utilities, and finally the image arrays were compiled into a numpy array, and their corresponding labels were also compiled into a numpy array. The labels were encoded using the one hot vector encoding technique. Finally, the dataset was split into a train data (70 percent of the original dataset) and test data (30 percent of the original dataset).

3.1.3 Model Architecture :

1. **Pretrained model :** The assignment requires us to fine-tune a pre-trained image classification model by modifying the pre-trained model's final fully connected layers. The pre-trained model used for this section of the assignment is the inceptionV3 image classification model, which is a powerful image classification deep learning model that is pretrained on the ImageNet dataset.

2. **Fine-tuning pre-trained model for garbage classification task :** The pretrained InceptionV3 model is utilized as a feature extractor, with its weights frozen to preserve the learned representations. The input shape is set to (299, 299, 3) to accommodate the size of the garbage images. The subsequent layers of the model include a GlobalAveragePooling2D layer, which is followed by a batch normalization laayer, followed by a dense layer with 1024 neurons and ReLU activation function to capture complex patterns in the data. The final dense layer consists of 6 neurons with a softmax activation function, aligning with the number of classes in the garbage classification task. This architecture leverages the knowledge learned by InceptionV3 while adapting the model to the specific task of distinguishing between different types of garbage. The inclusion of a dense layer allows the model to learn task-specific features and make accurate predictions for garbage classification.

3.1.4 Methodology

The fine-tuned model with the architecture described above was trained and validated for 20 epochs. The Adam optimizer was used with a learning rate of 0.0001, the loss selected for this task was the categorical cross entropy loss, and a batch size of 8 was used to group the samples into batches before they were fed into the model either for training or testing. The model was first trained on the training set and validated on the test dataset. During the training process, the model's validation loss metric was continuously monitored to obtain the parameters of the model when the validation loss was at the lowest possible value. This was done by saving the model parameters as a checkpoint when the validation loss of the model was at it's lowest possible value. This saved model was then evaluated / tested against the training set as well as the test set.

3.2 Results

In the second section of the assignment, an experiment was conducted to fine-tune a pretrained model to perform a specific image detection task - garbage classification task. The results of this experiment are described in this section.

The accuracy vs epoch curve for both training accuracy and validation accuracy is shown in Figure 4.

Evaluation on type of dataset	**Accuracy**
Test Dataset	0.571
Train Dataset	0.786

Table 4: Evaluation Results of Fine-tuned model

The table 4 shows the test accuracy scores and losses on the test dataset as well as on the train dataset.

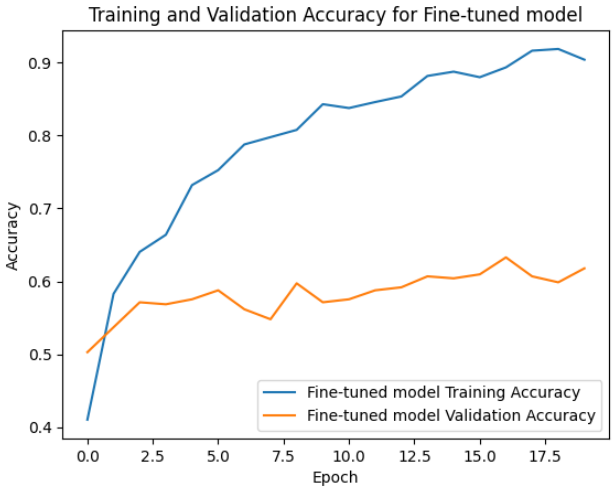


Figure 4: Accuracy vs Epoch curves for the fine-tuned model

3.3 Analysis

In this section, the results obtained in the previous sections are analysed. On assessing the results, the following observations were made.

Firstly, we can see that the fine-tuned model is clearly overfitted. As we can see from the Figure 4, the training accuracy is increasing, however the validation accuracy seems to be rising very slowly, which is a clear sign of overfitting. This could be due to a number of reasons, for instance the architecture of the model might be too complex for the given dataset. Another reason could be that the dataset is small or not representative of the entire population, and hence the model might memorize the training examples instead of learning general patterns.

Secondly, a number of different visual features were learned by the model. This was shown in the last few cells of the jupyter notebook code for question 2. The lower level layers learnt features such as edges and textures, As you move up the layers, the model learns to recognize basic shapes like circles, squares, and triangles. Intermediate layers capture parts of objects, such as body of a bottle, tag on carboard box etc. Finally, the higher layers could represent more complex object hierarchies, combining parts into complete objects. The model may learn to recognize complex patterns and compositions that are characteristic of certain objects, such as metal objects, cardboard boxes.