REPORT ON NATURAL SCENE CLASSIFICATION

Different libraries were imported, each of these libraries has a distinct role in creating and testing a neural network. These libraries make it easier to preprocess data, develop models, train, evaluate, and visualise my findings.

OS is used give access to features that are dependent on operating system such as reading or writing from the file system.

Shutil is used in advanced operations like copying and deleting files on an individual files and groups of files.

Itertools is used in enabling efficient looping and construction of complicated iterators by implementing several iterators building blocks.

Pathlib this library offers a filesystem path object-oriented interface. It makes the use and manipulation of filesystem paths easier.

PIL is used to open, edit, store a wide range of picture file formats.

CV2(OpenCV) is a library for computer vision task, use in supporting real-time image processing which includes functions of image and video capture, analysis and manipulations.

NumPy is a python module for scientific computing, used to support arrays, matrices and many wide ranges of mathematical functions.

Pandas is an effective library for data analysis and manipulation. It provides data structures like DataFrame which are necessary for data wrangling.

Matplotlib.pyplot(plt) is a python plotting package for making interactive, animated and static visualizations.

Seaborn(sns) is an elegant statistical graphic creator which is built with Matplotlib and offer a high-level interface.

Plotly.express(px) is used for creating interactive plots.

Sklearn (scikit-learn) is a machine learning library that includes simple and efficient tools for data mining and analysis. It is particularly used to divide data into training and testing sets. It computes confusion matrix to evaluate the accuracy of a classification. It builds a text report showing classification metrics. Its used in plotting confusion matrix using confusion-matrix-display.

TensorFlow is an open-source deep learning framework by google, used for machine learning and neural network models and for variety of functions.

Kera’s is an API for building and training a deep learning model. It provides easy to use neural network building pieces which has become a part of TensorFlow.

ImageDataGenerator is used for generating batches of tensor images data with real-time data augmentation.

Sequential is a keras model with a recurrent layer stacking.

Layers (Dense, Conv2D, MaxPooling2D, Flatten, Activation, Dropout, BatchNormalization) this are building blocks of neural network, assigned for a specific function to each type of layer.

Optimizers (Adam, Adamax) are algorithms that use the training data to iteratively update the network weights.

Regularizers are functions used during optimisations to add penalties on layer parameters to help in preventing overfitting.

Pretrained models (Xception, VGG16, ResNet50 ) these are pre-trained models that have been trained on a large datasets such as ImageNet.

Warnings is a standard library for displaying warning messages. It helps in alerting the user of any potential issues with the code. The “filterwarnings (‘ignore’)” line suppresses this.

from google.colab import drive,( drive.mount('/content/drive'): this line is used to import my drive from google colab. The mount and the content/drive function are used to authorize google colab to access my drive, which give it access to the drive.

Extract\_Dir is used to specify where the zipped file is located, where to extract content from.

PART A

The code provided below implements a neural network for classifying natural scenes using the EfficientNetB0 model as the base. The model is fine-tuned with additional layers to enhance its classification capabilities.

Model Details and Critical Evaluation

Model Architecture and Rationale

The modern convolutional neural network EfficientNetB0, which is renowned for its accuracy and efficiency in image classification tasks, is the foundation of the model architecture that was chosen. EfficientNetB0 was selected because it offers a balanced trade-off between performance and model size, making it appropriate for jobs involving the of natural scenes classification.

Model Layers:

Base Model (EfficientNetB0):

EfficientNetB0 has strong feature extraction capabilities because it has been pre-trained on the ImageNet dataset. Using a pre-trained model facilitates the use of transfer learning, which increases accuracy and speeds convergence.

Include Top: False: This allows us to customize the classification head for our specific task.

Batch Normalization: this is added to stabilize and accelerate the training process by normalizing the inputs of each layer. This helps in reducing internal covariate shift, leading to faster convergence.

Dense Layers (2 x 256 units): To introduce non-linearity to help with convoluted feature learning, two dense layers, each consisting of 256 units, are used. A choice was made while selecting 256 units between computing performance and model complexity. Two layers make it easier to see more complex patterns in the data that one layer would overlook. Overfitting may result from using more than two layers, particularly when working with a small dataset.

Output Layer (Dense with SoftMax Activation): The last dense layer outputs the probability for each class using a SoftMax activation function. The number of classes in the dataset is equal to the number of units

Hyperparameters

Learning Rate (0.001): The Adamax optimiser is set to a learning rate of 0.001, which strikes a balance between update stability and learning velocity. Adamax works well with sparse gradients and is resilient to noisy gradients.

Loss Function (Categorical Crossentropy):

categorical crossentropy compares the projected probability distribution to the actual distribution, it is appropriate for multi-class classification applications.

The Rationale:

Batch Normalization: It is being proven that adding batch normalisation layers increases the model's stability and rate of convergence during training. The gradual decrease in loss and rise in precision over epochs is the result of this.

Two Dense Layers: The choice of using two deep layer selection aids in capturing intricate details without overfitting. Three dense layers and one dense layer were tested against this setup. It improves better validation accuracy and reduced loss were attained by the two-layered dense layer model, suggesting a good compromise between underfitting and overfitting.

Learning Rate and Optimizer: It was discovered through experimentation that the Adamax optimiser, using a learning rate of 0.001, offers a suitable trade-off between stability and convergence speed. While lower learning rates significantly slowed down convergence, higher learning rates resulted in unstable training.

Results:

Training Performance:

Final Training Accuracy: 86%

Final Training Loss: 2.12

Validation Performance:

Final Validation Accuracy: 98%

Final Validation Loss: 0.056

Test Performance:

Test Accuracy: 98%

Test Loss: 0.037

Good generalisation from the training data is demonstrated by the model's performance on the test set. The model's capacity to accurately categorise the natural scenes, with excellent precision, recall, and F1-scores across most classes, is further demonstrated by the confusion matrix and classification report.

Critical Evaluation:

The chosen model has proven to perform well in classifying natural events, as its own arrangement. Experimental validation provides justification for the selection of EfficientNetB0 as the basic model in combination with batch normalisation and two dense layers. The model is appropriate for real-world uses in natural scene classification because of its strong convergence tendency and generalisation capacity.

The model's strengths as well as potential areas for improvement are highlighted by the assessment metrics and visualisations, which offer thorough insights into the model's performance. All things considered, the model's architecture and hyperparameter choices are sound, producing a fair and useful model for the task in question.

PART B

Report on the Impact of Label Manipulation

Experiment Setup

The objective of this experiment is to assess the impact of label manipulation on the performance of the neural network model developed for natural scene classification. We consider various percentages of label flipping (5%, 10%, 15%) and observe the performance degradation.

Methodology:

Label Flipping: In the training dataset, labels are flipped at certain percentages at random. Every label will be swapped into a new one that is selected at random.

Training and Evaluation The altered labels are used to retrain the model, and the validation set is used to assess how well it performs. Metrics for accuracy and loss will be provided.

Visualization: Plots are used to illustrate the effects on different label flipping percentages on the accuracy and loss of the model.

Results Analysis

The results are summarized in the plots, showing validation accuracy and loss for different percentages of flipped labels over the epochs.

Observations:

5% Flipped Labels:

Validation accuracy is slightly less than for the model trained using original labels.

A slight increase in the validation loss signifies an imminent decrease in the model's performance.

Training Performance:

Final Training Accuracy: 99%

Final Training Loss: 0.02

Validation Performance:

Final Validation Accuracy: 95%

Final Validation Loss: 0.412

Test Performance:

Test Accuracy: 92%

Test Loss: 0.638

10% Flipped Labels:

The validation accuracy decreases even more, exhibiting a more substantial impact on the model's generalisation. There is a more obvious increase in the validation loss, which shows more confusion throughout training.

Training Performance:

Final Training Accuracy: 89%

Final Training Loss: 1.378

Validation Performance:

Final Validation Accuracy: 99%

Final Validation Loss: 0.021

Test Performance:

Test Accuracy: 99%

Test Loss: 0.022

15% Flipped Labels:

A significant reduction in validation accuracy shows a serious problem with the model's performance.

The increasing validation loss indicates a greater percentage of model misclassification.

Training Performance:

Final Training Accuracy: 99%

Final Training Loss: 0.025

Validation Performance:

Final Validation Accuracy: 83%

Final Validation Loss: 1.425

Test Performance:

Test Accuracy: 85%

Test Loss: 1.306

Conclusions:

• Even at low percentages, label flipping decreases the performance of the model. As the percentage of flipped labels improves, the effect becomes more noticeable.

• The model's validation accuracy and loss metrics deteriorate significantly with increased label manipulation, demonstrating the importance of label integrity in training data.

• This experiment illustrates the requirement for robust data validation methods and shows how susceptible neural networks are to label noise.

Recommendations:

To mitigate the impact of label manipulation, the following strategies can be considered:

Data Validation: Prior to training, implement rigorous processes for data validation to identify and fix samples that have been inaccurately labelled.

Robust Training: To increase the model's resistance to label noise, use robust methods of training including regularisation approaches and noise-tolerant loss functions.

Anomaly Detection: Establish systems for detecting anomalies in training data to identify suspicious trends that might point to possible label manipulation.

These techniques can be used to protect the training data's integrity, assuring accurate and dependable model performance.

PART C

Theoretical Analysis of Detecting Label Manipulation in Neural Network Training Pipelines

Introduction

The process of manipulating labels within a training dataset can significantly impact the neural network model's performance. This review looks at ways to recognise and counter these manipulations, combining information gathered from the literature and suggesting new methods to improve the neural network training approach.

Literature Review

Noise-Robust Loss Functions:

Categorical Cross-Entropy with Regularization: It is possible to change loss functions to make them more resistant to label noise. The impact of inaccurate labels might be lessened by using strategies like label smoothing, which disintegrates a tiny portion of the right label probability among other classes.

Bootstrapping Loss: By modifying the loss function to take into consideration a weighted combination of the specified labels and the model's predictions, this approach effectively minimises the effect of noisy labels.

Data Validation Techniques:

Outlier Detection: To find outliers in the dataset, one can use machine learning models and statistical approaches. Samples with suspect label distributions can be diagnosed with the aid of techniques such as One-Class SVM and Isolation Forests.

Human-in-the-Loop Verification: To guarantee label correctness, a human-in-the-loop approach might involve frequent manual verification of samples collected at random from the dataset.

Robust Training Approaches:

Noise-Tolerant Training: Approaches like Mentor Net are shown promising for handling label noise. Mentor Net uses a pre-trained model to filter out potential mislabelled samples and then uses that model to guide the training of the main model.

Semi-Supervised Learning: During training, combining labelled and unlabelled data could help increase robustness to label noise. Pseudo-labelling and consistency regularisation are two techniques that improve model performance in noisy environments.

Novel Detection Strategies

Entropy-Based Label Validation:

Concept: The uncertainty in the model's predictions is measured by entropy. The entropy of predictions for every training sample can be used to determine when the model is very uncertain, which may be a sign of label manipulation.

Implementation: Estimate the SoftMax output's entropy for every training sample. Samples that have entropy values are noticeably greater than the mean, which may be marked for additional study.

Model Agreement Check:

Concept: Train multiple models (e.g., ensemble methods) on the same dataset and compare their predictions. Consistently high disagreement between models on certain samples could indicate mislabelled data.

Implementation: After training a group of models, determine the variance of each sample's predictions. High prediction variance samples may be identified as possibly inaccurately labelled.

Clustering-Based Label Verification:

Concept: Employ clustering methods to create feature-based groups of related samples. Examine the labels within every cluster and look for discrepancies.

Implementation: Apply clustering techniques to the sample feature representations (K-Means, DBSCAN, etc.). Determine the label distribution throughout each cluster. Label manipulation may be shown by clusters with non-uniform label distributions.

Proposed Methodology

Our suggestion is a multifaceted strategy that combines the methods mentioned above to identify label manipulation.

Pre-Training Data Validation:

Prior to training, use clustering-based label verification to find any potential inconsistencies in the dataset.

Apply human-in-the-loop verification to samples that have been flagged.

During Training:

In order to maintain the uncertainty of the model's predictions, use entropy-based label validation.

To reduce the effect of noisy labels, use loss functions that are adaptable against noise, such as categorical cross-entropy with label smoothing.

Post-Training Analysis:

To find samples with large prediction variance, use an ensemble of models to perform model agreement checks.

Re-examine samples that were flagged and, if required, carry conduct more human verification.

Findings and Conclusion

Effectiveness of Detection:

A comprehensive strategy for identifying label manipulation is offered by the combination of model agreement tests, clustering-based verification, and entropy-based validation.

When combined, these techniques increase the probability of finding samples that have been incorrectly labelled and guarantee the integrity of the dataset.

Implementation Considerations:

For manual verification, the suggested approaches involve more computational power and effort from humans. Better model robustness and reliability, however, make the investment worthwhile.

Future Directions:

Automated techniques for incorporating these strategies into the training pipeline can be examined in more detail.

To enhance the identification of label manipulation, more sophisticated methods such deep anomaly detection models could be researched.

To sum everything up, the performance and dependability of neural network models depend on the detection and mitigation of label manipulation. The multifaceted strategy that has been offered, which integrates both new and traditional methodologies, offers a strong foundation to tackle this difficulty.