DS- Project Group 53 submission

Edition: 2023 2A
Project: CIRI
Primary Topic: CV&IC
Secondary Topic:
Members: .. Caitlin Wong Yu-Lin, Giulia Pais
Last update: 25/04/2024, 10:02:19

Motivation

Amidst the rising frequency and intensity of natural disasters and other critical incidents, the need for efficient and effective response strategies has never been more pressing. Emergency response teams and government agencies tasked with managing such crises face immense challenges in accurately assessing the nature and severity of incidents to allocate resources promptly and effectively. Traditional methods rely on manual identification and understanding of incidents through on-the-ground responders and analysts, falling short in providing timely and precise information, leading to delays in response efforts and potentially exacerbating the impact of disasters.

Therefore, it is both relevant and important to leverage on image classification technology as a tool for improving incident detection and response. With the widespread use of social media platforms like *Instagram* and *X*, where users typically post geo-tagged images showing their current surroundings and potentially incidents in their immediate vicinity, these platforms present large databases of images which can be used for analysis. Harnessing the power of machine learning algorithms, in combination with images extracted from social media posts, has the potential to vastly improve early warning systems for disasters and accidents. This would be particularly useful for emergency response teams and government agencies, who could benefit immensely from the ability to swiftly detect and monitor incidents such as floods, earthquakes, and car accidents, enabling them to proactively implement mitigation measures and allocate resources where they are most needed.

Examples of non-governmental stakeholders include management teams from - the United Nations Office for the Coordination of Humanitarian Affairs (OCHA), who coordinate responses to emergencies and relief efforts; *Instagram* and *X*, which would be able to play a pivotal role in saving lives by collaboration with relief organisations in implementation of image classification technology for incident detection.

(Business/Research) questions

The central inquiry driving this study is: "To what extent can image classification technology aid in early warning systems for natural disasters, and what are the potential benefits and limitations of its implementation?" This question is formulated following the SMART criteria, ensuring clarity, measurability, achievability, relevance, and timeliness.

Specificity: the research question focuses on the role of image classification technology in enhancing early warning systems for natural disasters and the extent to which this technology can contribute to improving incident detection and response processes.

Measurability: the question implies measurable outcomes, such as the accuracy rates of image classification algorithms in detecting incidents and the identification of specific benefits and limitations associated with the implementation of image classification technology.

Achievability: this question is achievable through readily available image datasets and experimentation with parameters of existing CNN architectures optimised for image classification. It seeks to provide evidence-based insights into the feasibility and effectiveness of integrating image classification technology into disaster management frameworks.

Relevance: the question addresses a critical need in the field of disaster management and public safety, aligning with broader efforts to enhance preparedness, response, and recovery efforts in the face of natural disasters. By examining the potential benefits and limitations of image classification technology, this research contributes to advancing knowledge and informing decision-making processes in disaster management.

Timeliness: the increasing frequency and severity of natural disasters worldwide calls for exploration of innovative technologies such as image classification. This research aims to provide timely insights into the role of image classification technology in improving early warning systems, thereby facilitating more effective responses to natural disasters and ultimately saving lives and resources.

By addressing this research question, the project endeavours to offer valuable guidance for policymakers, emergency responders, and technologists striving to mitigate the impact of disasters and accidents on communities worldwide.

Source data

Introduction to Data Quality and Initial Observations

We conducted a detailed inspection of the Incidents-subset dataset. During this initial analysis, it became apparent that the classes in the dataset naturally grouped into three tiers based on sample abundance:

- Adequately represented classes: classes like "airplane accident," "earthquake," "car accident," and "flooded" had sample counts ranging from 872 to 966.
- Moderately under-represented classes: classes such as "collapsed," "ice storm," "wildfire," and "volcanic eruption" had sample counts ranging from 615 to 688.
- Severely under-represented classes: classes including "tornado," "nuclear explosion," "bicycle accident," and "oil spill" had sample counts ranging from 231 to 293.

During this phase, we also encountered data integrity issues, such as 7 corrupted images and some mislabeling errors upon visual inspection of repeated random samples from the dataset.

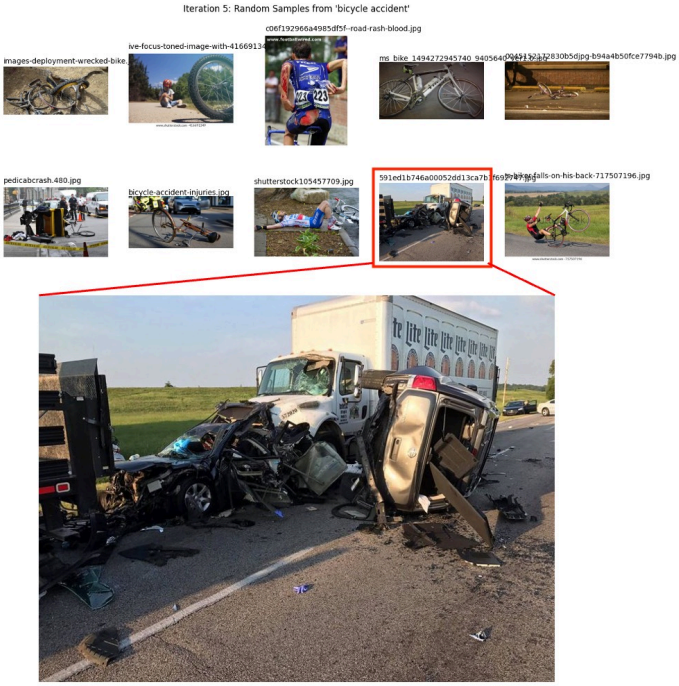


Figure 1. *File 591ed1b746a00052dd13ca7b1f692747.jpg labelled "bicycle accident", but should be labelled "car accident".*

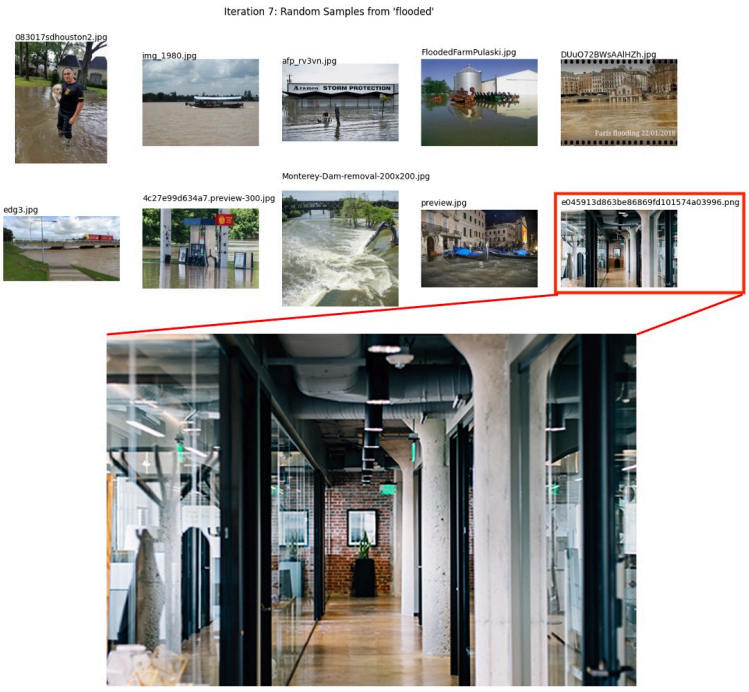


Figure 2. *File e045913d863be86869fd101574a03996.png labelled "flooded", when it does not depict any relevant incident.*

Addressing Data Quality Issues

To address the imbalance across various classes in the dataset, we developed a tiered augmentation strategy using advanced techniques from PyTorch's torchvision package. Each augmentation level built upon the previous one, ensuring a progressive increase in data diversity and complexity:

- Base augmentation for adequately represented classes: we utilized **AutoAugment** [1], which automatically applies a series of transformations that have been optimized through learning from large datasets like ImageNet. This base level included adjustments such as rotations, color enhancements, and flips, providing a foundational increase in image variability.
- Intermediate augmentation for moderately under-represented classes: building on the base augmentation, **RandAugment** [2] was added for classes requiring additional diversity. RandAugment selects transformations at random, reducing the need for pre-determined policies and adding an additional layer of randomness and complexity to the images.
- Heavy augmentation for severely under-represented classes: for the most under-represented classes, **AugMix** [3] was incorporated in addition to the previous augmentations. This method blends multiple augmentation techniques to create images that maintain their naturalistic appearance while introducing complex visual perturbations.

These strategies were chosen to not only balance the dataset but also to introduce variations that help in building a model resilient to different visual perturbations. Visual examples of these transformations can be found in Appendix section I.

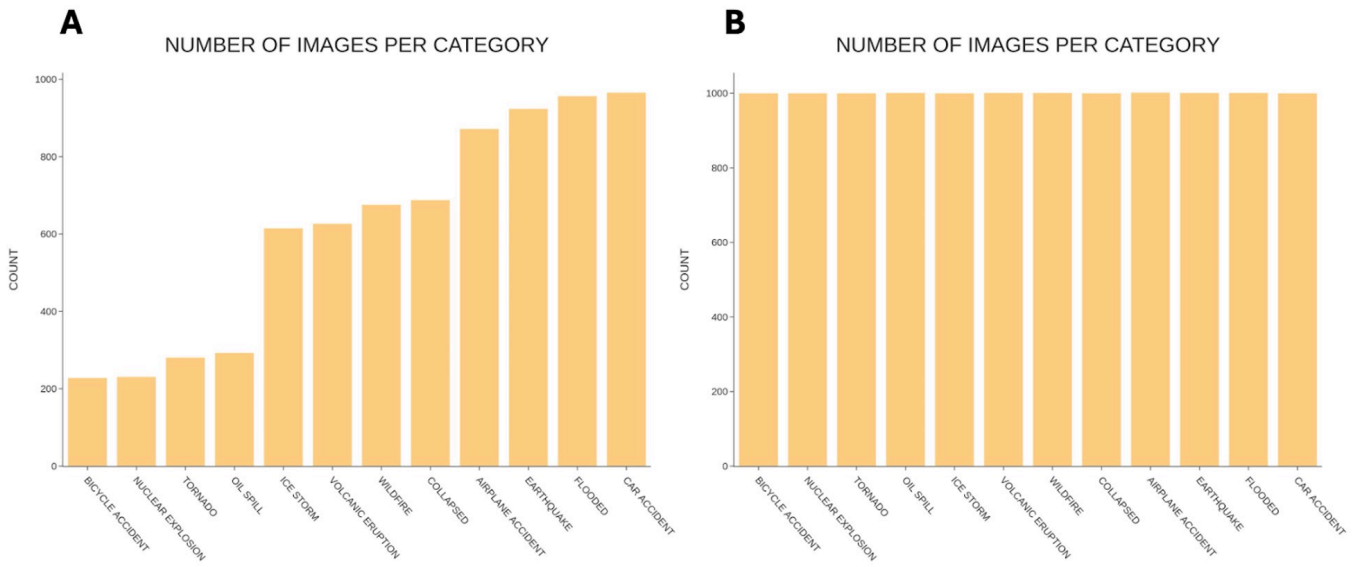


Figure 3. *Overview of category abundances A) before data augmentation and B) after data augmentation.*

To manage corrupted files, we implemented a custom batch collate function in the DataLoader, which allowed us to skip these files without interrupting the loading process.

Due to the substantial size and complexity of the Incidents-subset dataset, manually correcting mislabeled samples was deemed impractical and resource-intensive. The subjectivity involved in labeling, coupled with the risk of introducing additional errors, made manual correction unfeasible. Furthermore, research suggests that machine learning models can tolerate a certain level of label noise without a significant drop in performance [4]. Thus, focusing on other areas of the project that provided greater benefits in model accuracy and robustness was prioritized over correcting these labels.

Method

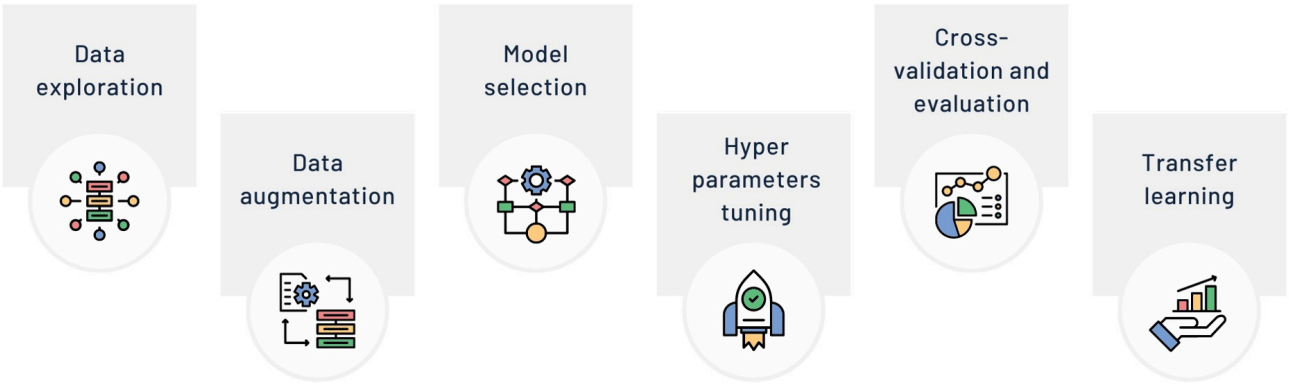


Figure 4. *Graphical representation of all the steps followed in our methodology.*

Limitations
Our approach was constrained by limited access to Google Colaboratory's GPUs, with strict usage time limits, and tight project deadlines that restricted extensive experimentation and tuning.

Model selection
We chose ResNet50 based on literature recommendations [5] and its efficacy in processing complex images. Wide ResNet50 [6] was selected for its enhanced accuracy without a substantial increase in computational demands. Vision Transformers [7] were excluded due to their high resource requirements and lengthy training times.

The adoption of established network architectures over developing new models was driven by the following:

- Designing efficient and effective CNNs from scratch is a complex endeavor that requires deep expertise
- Established models were extensively tested and optimized across various datasets and scenarios
- We can establish a benchmark and any necessary adjustments or enhancements can be more effectively planned and implemented on top

We adapted both models to work with our data by substituting the last fully connected layer to match the number of categories in our dataset. Both models were trained using *AdamW* optimizer, *ReduceLROnPlateau* scheduler and cross entropy as loss function.

Images representation
We developed a custom ***CIRI_Dataset*** class, leveraging PyTorch's ***Dataset*** and ***DataLoader*** for handling image data. This setup processes images into tensors of shape 3 x W x H (224×224 pixels, RGB), applying model-specific transformations to ensure compatibility with the neural network's input layer. Transformation details are provided in Appendix section II.

Hyperparameter tuning
To optimize hyperparameters efficiently within our computational limits, we utilized nested cross-validation with three inner and five outer folds. We focused on tuning epochs, batch size, and learning rate through a randomized search, limiting our search to five options per fold for feasibility. Due to high computational demands and in light of several unsuccessful attempts, we conducted tuning on a 20% dataset sample, recognizing that this may not fully represent the entire dataset's nuances and could affect model performance. Hyperparameter selections were based on accuracy.

Cross-validation and evaluation
We conducted a comprehensive evaluation using a full 5-fold cross-validation on the entire dataset using the best hyperparameters.
Using the entire dataset for this phase of evaluation allowed us to assess the models' performance more accurately and reliably. It enabled us to understand how well the models generalize to new, unseen data, providing a robust measure of model effectiveness across different subsets of data.

To evaluate the performance of each model comprehensively, we utilized accuracy, precision, recall and F1-score.

Transfer learning

We utilized pre-trained ImageNet models for our project, modifying and retraining the last fully connected layer to align with our dataset's class categories. This basic form of transfer learning (freezing all but the last layer) helps tailor the model to our specific needs. We evaluated model performance using accuracy, precision, recall, and F1 score.

The process can be fine-tuned by progressively unfreezing layers of the architecture to re-learn the corresponding weights. Unfortunately, our attempts to implement progressive unfreezing were not successful, as we were unable to complete the training process with 2 unfreezed layers due to the limitations already mentioned.

Results

Hyperparameter tuning
The optimal hyperparameter configurations identified for our models, based on the best accuracy achieved, are listed below. More detailed results are included in Appendix section III.

ResNet50: best accuracy 39%
- Epochs: 20
- Batch size: 32
- Learning rate: 0.0006769998458972

Wide ResNet50: best accuracy 37%
- Epochs: 5
- Batch size: 32
- Learning rate: 0.0001077774458011

These configurations, while the best found under our constraints, are potentially sub-optimal due to the limited amount of data and the computational resources available. We acknowledge the necessity for more refined tuning to ideally enhance these models' performance.

Cross validation and model performance
During 5-fold cross-validation, both models displayed distinct patterns indicative of their fit:

- ResNet50: learning curves suggest overfitting (**Figure 5**); training loss decreases consistently, whereas test loss starts to plateau or rise after an initial decline, particularly noticeable after the 10th epoch. This discrepancy between training and test accuracy indicates that the model memorizes rather than generalizes from the training data.
- Wide ResNet50: exhibits clear signs of underfitting (**Figure 6**), where both training and test losses are high, and accuracies remain uniformly low across epochs. This behavior suggests that the model lacks the capacity to learn the necessary features from the data provided, or the training is insufficient.

There may be several reasons why our models did not perform well:
- The models are too complex with respect to the given dataset size and this hampers the ability to generalise properly - this is especially true for ResNet50: simply stopping the training process at epoch 10 wouldn't benefit us, as the model would be underfitting with a train accuracy around 60% and a test accuracy even lower.
- The data lacks variety or has many similar examples - this might stem from the fact that we performed extensive data augmentation, in particular for some of the most under-represented classes
- The combination of hyperparameters chosen could be sub-optimal - very likely since we had limited computational power and we tested only 5 combinations per each fold and for a limited choice of tunable parameters among many

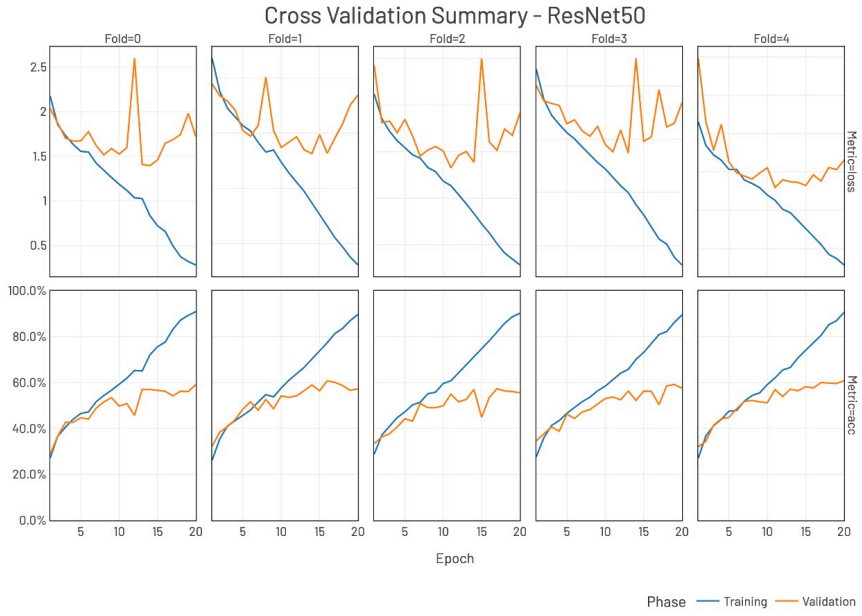


Figure 5. *Overview of the 5-fold cross validation process on ResNet50. Plotted in the first row, for each separate fold are the curves for training and validation cross entropy loss for each epoch; plotted on the second row the curves for the training and validation accuracy for each epoch. We can observe that the general trend is very similar in all the folds, suggesting that performance does not depend on a particular split of the dataset, and all folds indicate clearly the presence of overfitting.*

Figure 6. *Overview of the 5-fold cross validation process on Wide ResNet50. Plotted in the first row, for each separate fold are the curves for training and validation cross entropy loss for each epoch; plotted on the second row the curves for the training and validation accuracy for each epoch. Some data points are missing for fold 3, presumably due to errors when persisting the result files. The model shows clear signs of underfitting.*

epoch	loss	accuracy	precision	recall	f1
0	2.9803503	32.0%	39.2%	32.0%	28.2%
1	2.1541555	34.5%	37.0%	34.5%	32.0%
2	1.7815018	41.4%	46.5%	41.4%	40.5%
3	2.1180958	44.2%	47.3%	44.2%	42.9%
4	1.6293234	44.8%	46.6%	44.8%	42.3%
5	1.4894065	48.6%	51.5%	48.6%	46.5%
6	1.4462754	51.7%	52.8%	51.7%	50.6%
7	1.4089058	52.2%	54.4%	52.2%	51.4%
8	1.4849703	51.6%	56.6%	51.6%	50.8%
9	1.5549524	51.2%	54.9%	51.2%	49.8%
10	1.2971787	56.9%	57.1%	56.9%	55.6%
11	1.3973336	53.9%	58.0%	53.9%	52.5%
12	1.3736678	57.1%	59.3%	57.1%	56.6%
13	1.3639382	56.4%	59.5%	56.4%	55.9%
14	1.3236651	58.2%	60.7%	58.2%	58.1%
15	1.4634603	57.6%	60.5%	57.6%	57.6%
16	1.3816664	60.0%	63.6%	60.0%	60.6%
17	1.5582394	59.8%	60.8%	59.8%	59.4%
18	1.5297266	59.6%	61.7%	59.6%	58.3%
19	1.6545777	60.8%	62.9%	60.8%	60.4%

Table 1. *Training progress and metrics for the best performing fold (4) of ResNet50.*

epoch	loss	accuracy	precision	recall	f1
0	1.8844	37.8%	40.0%	37.8%	35.6%
1	1.7571	40.3%	43.5%	40.3%	39.3%
2	1.5903	44.8%	48.0%	44.8%	44.6%
3	1.6001	45.5%	51.6%	45.5%	43.2%
4	1.4539	51.2%	55.1%	51.2%	50.5%

Table 2. *Training progress and metrics for the best performing fold (4) of Wide ResNet50.*

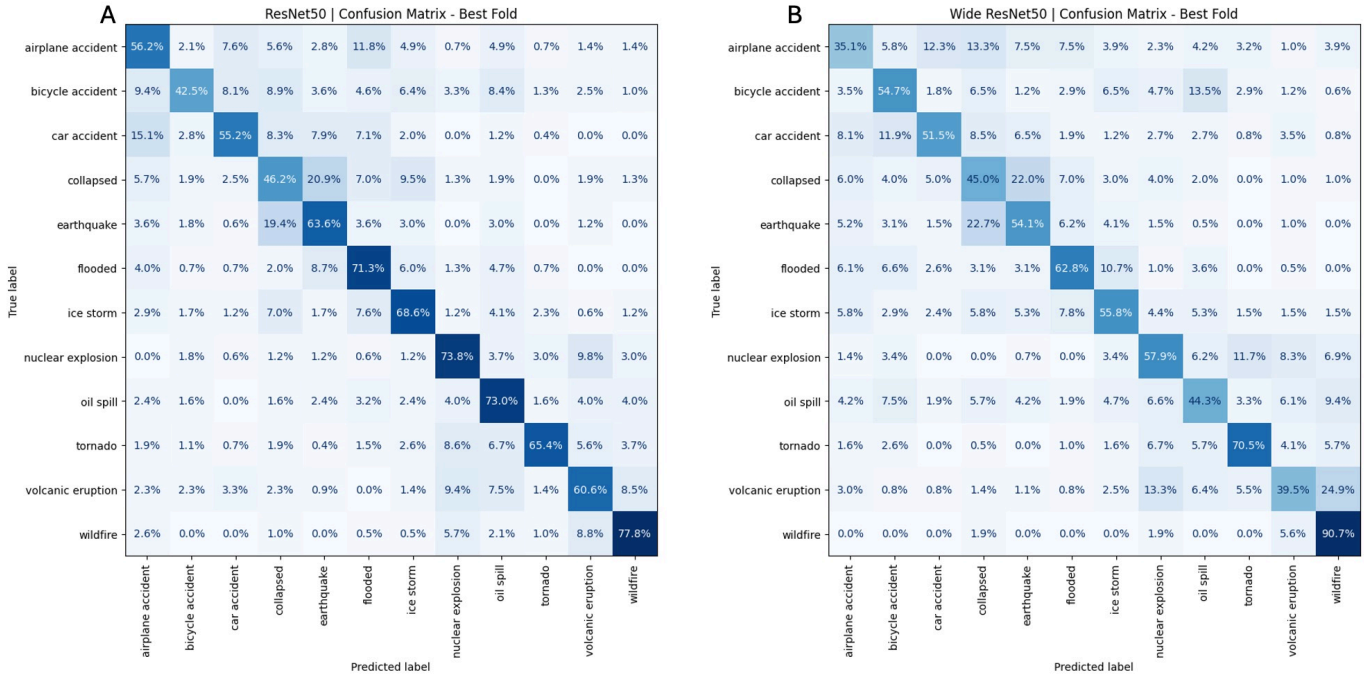


Figure 7. *Confusion matrices for the best folds based on maximum accuracy for ResNet50 (**A**) and Wide ResNet50 (**B**). In A we can observe that some classes have a lower accuracy, but counter-intuitively, this does not interest solely the original under-represented classes but also well-represented ones like "car accident". This might suggest lack of variety of the presence of mislabelled samples for that category.*

Transfer learning

Due to our limitations, we performed the baseline form of transfer learning for 5 epochs for both models leaving batch size and learning rate unchanged from previous experiments. The performance in terms of accuracy (as well as precision, recall and f1-score) greatly improved, as expected and observing the metrics reported for each epoch in **Table 3**, we can say the model is not overfitting or underfitting. As previously mentioned, unfortunately data for the fine tuning with layer 4 of both architectures unfreezed is unavailable because it wasn't possible to complete the training process. From the confusion matrices in **Figure 8** we can observe that transfer learning greatly reduced the misclassification rate for all classes, but in particular classes like "bicycle accident" or "collapsed" that had many misclassification errors in the previous experiments.

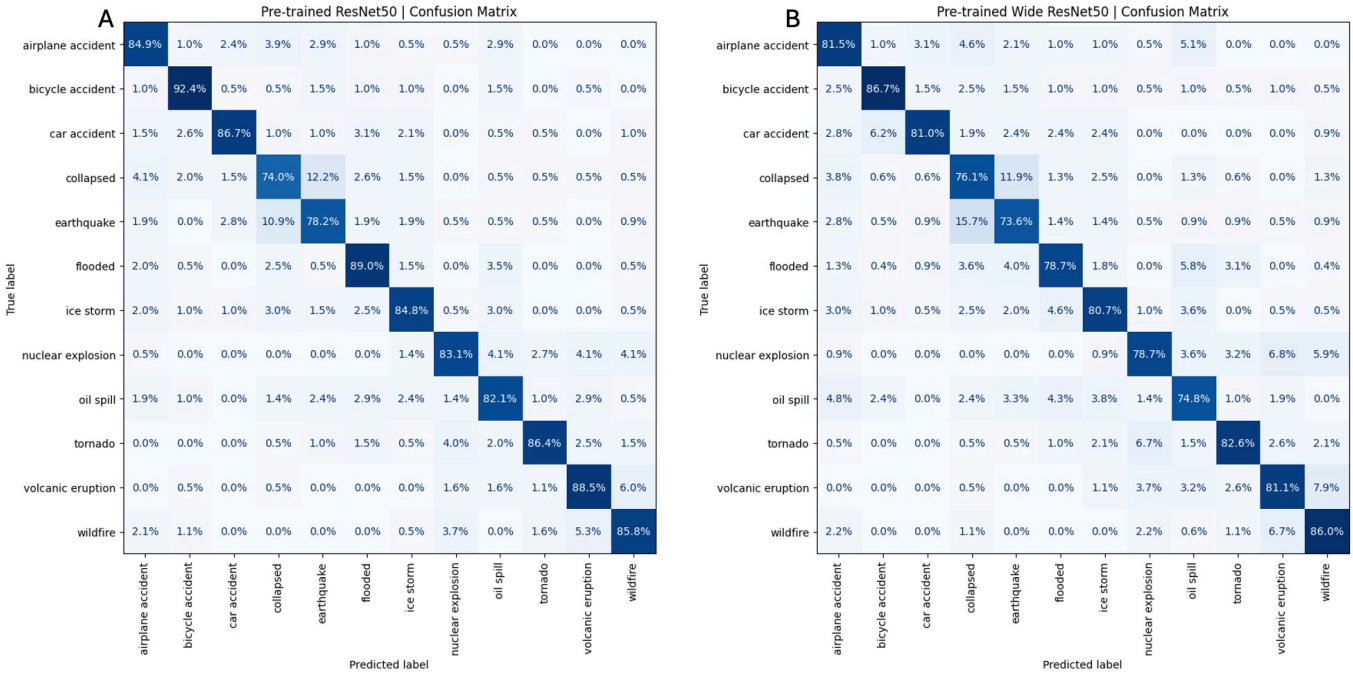


Figure 8. *Confusion matrices for the pre-trained ResNet50 (**A**) and Wide ResNet50 (**B**). Accuracy of all classes improved quite significantly, especially classes like "bicycle accident" and "car accident".*

A

epoch	train_loss	train_acc	val_loss	val_acc
1	1.2081957	69.97%	0.802462	79.38%
2	0.6782059	81.01%	0.6231183	82.25%
3	0.5541031	84.24%	0.5768036	83.46%
4	0.4879313	85.86%	0.5369877	83.88%
5	0.4450718	87.52%	0.4997768	84.58%

B

epoch	train_loss	train_acc	val_loss	val_acc
1	1.919728128	55.26%	1.546452427	72.38%
2	1.274005464	75.25%	1.175788415	75.96%
3	0.996240142	78.40%	0.972180707	78.17%
4	0.858152467	79.78%	0.867084484	79.17%
5	0.762740894	81.19%	0.786820802	80.04%

Table 3. *Training progress for the transfer learning process on ResNet50 (**A**) and Wide ResNet50 (**B**).*

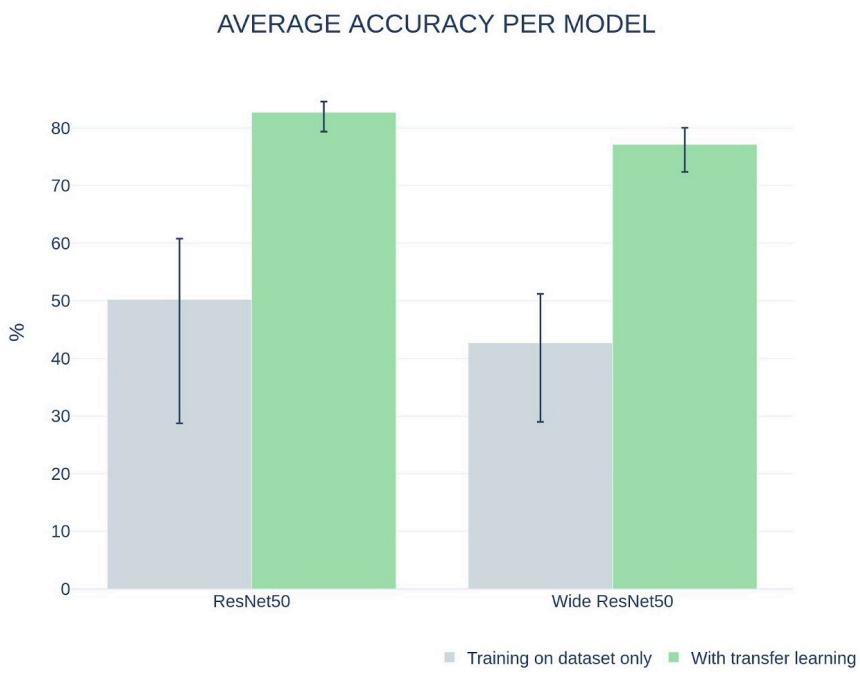


Figure 9. *Comparison side by side of the accuracy of both models without transfer learning (in grey) and with transfer learning (in green). The reported accuracy for the no-transfer learning approach is the average on 5 folds, with the error bars indicating the minimum and maximum respectively. The reported accuracy for the transfer-learning approach is the average over all epochs, with error bars indicating minimum and maximum respectively.*

Reliability of results

Limitations of the Study

Understanding the limitations inherent in our project is crucial for interpreting the results accurately. Key limitations include:

- Data constraints: the limited size and variability of our dataset, especially after intensive data augmentation for underrepresented classes, may have impacted the generalizability of our models. While augmentation helps in addressing class imbalance, it can also introduce biases if not properly managed.
- Computational resources: the restrictions on computational resources, particularly the intermittent access to GPUs and time limitations on Google Colaboratory, constrained our ability to execute extensive model training and hyperparameter optimization. This limitation potentially affected the depth of our training and the breadth of hyperparameter exploration.
- Model complexity: the complexity of models like ResNet50 and Wide ResNet50, while beneficial for handling large and diverse datasets, may not be entirely suitable for smaller, less varied datasets. This mismatch can lead to overfitting, where the model performs well on training data but less so on unseen data.

Measures to Ensure Reliability

To enhance the reliability of our findings, several measures were implemented:

- Nested cross-validation and k-fold cross validation: employing cross-validation approaches during hyperparameter tuning and model training provided a more robust estimate of the model's performance on unseen data and helped us identifying issues with our models
- Conservative interpretation of results: we have maintained a cautious approach in interpreting the performance metrics, considering the potential overfitting and underfitting issues indicated by our analysis. This approach helps in setting realistic expectations about the applicability and effectiveness of the developed models.

Conclusions & recommendations

Stakeholder

Our primary stakeholder is the United Nations Office for the Coordination of Humanitarian Affairs (OCHA). While executives of social media companies like Instagram and X are clear choices, organisations like OCHA would need to be invested in using this incident detection tool and have the influence to subsequently extend partnership to social media platforms.

Main Conclusions

Our analysis yielded several key insights:

- Model suitability: ResNet50 and Wide ResNet50, while robust for large datasets, encountered challenges when adapted to smaller, less diverse datasets. ResNet50 tended to overfit, indicating sensitivity to the limited data variety, while Wide ResNet50 underperformed, suggesting underfitting issues.

- Effectiveness of transfer learning: using transfer learning proved to be effective, significantly enhancing model performance. This underscores the utility of leveraging pre-trained models to improve learning efficiency and outcome in constrained environments.
- Impact of Computational Limitations: The constraints on computational resources significantly shaped our experimental scope and outcomes. This was evident in the limited hyperparameter tuning and the necessity to use a reduced data sample for tuning purposes.

Recommendations
Based on these conclusions, we recommend the following strategies:

- Alternative or complementary approaches to handle class imbalance: relying on sole augmentation might not be sufficient and might cause overfitting as we observed in our models. Such approaches may include integrating images from external datasets, generating new synthetic examples with the help of GANs (Generative Adversarial Networks) and oversampling the minority classes.
- Expanded hyperparameter exploration: with more resources, a broader and more thorough hyperparameter optimization should be undertaken. Techniques such as grid search could yield more optimal settings.
- Optimization of transfer learning: continue to utilize and optimize transfer learning strategies. Exploring advanced techniques like progressive layer unfreezing could allow for more nuanced adaptation to specific dataset characteristics.

Future research
Future studies could benefit from employing ensemble learning to improve accuracy and explore non-CNN models like Vision Transformers. Incorporating object detection before classification might also simplify and enhance the task.

Reflection

Addressing Project Challenges
Throughout this project, several challenges including computational limitations, model overfitting, and data diversity were addressed using strategic approaches such as transfer learning and hyperparameter tuning. These methods allowed us to optimize the available resources and improve model performance within the constraints imposed by our hardware and dataset size.

Relevance of Course Skills and Techniques
The skills and techniques learned during the course proved invaluable for applying theoretical knowledge to this real-world project. Concepts such as cross-validation, transfer learning, and data augmentation directly influenced our approach, enabling a practical understanding of how machine learning models can be adapted and optimized for specific tasks.

Additional Knowledge and Skills
While the foundation provided by the course was instrumental, additional knowledge in areas such as advanced model tuning, computational resource management, and ensemble methods could further enhance project outcomes. Skills in managing more extensive computational resources or using cloud computing platforms might allow for more extensive experimentation with larger datasets and more complex models.

Use of ChatGPT
For this project, ChatGPT was used to help with spell checking and summarization of concepts in this report, recommendation of scientific literature and code debugging.

ML issues

Class imbalance, computational limits and performance issues were all addressed in the previous sections.

Generalisation capabilities of the method

The model performance has been assessed through 5-fold cross validation, monitoring both cross entropy loss per epoch and accuracy per epoch. Nested cross-validation was used for hyperparameter tuning. Other details are mentioned in the "Methods" section and the "Results" section.

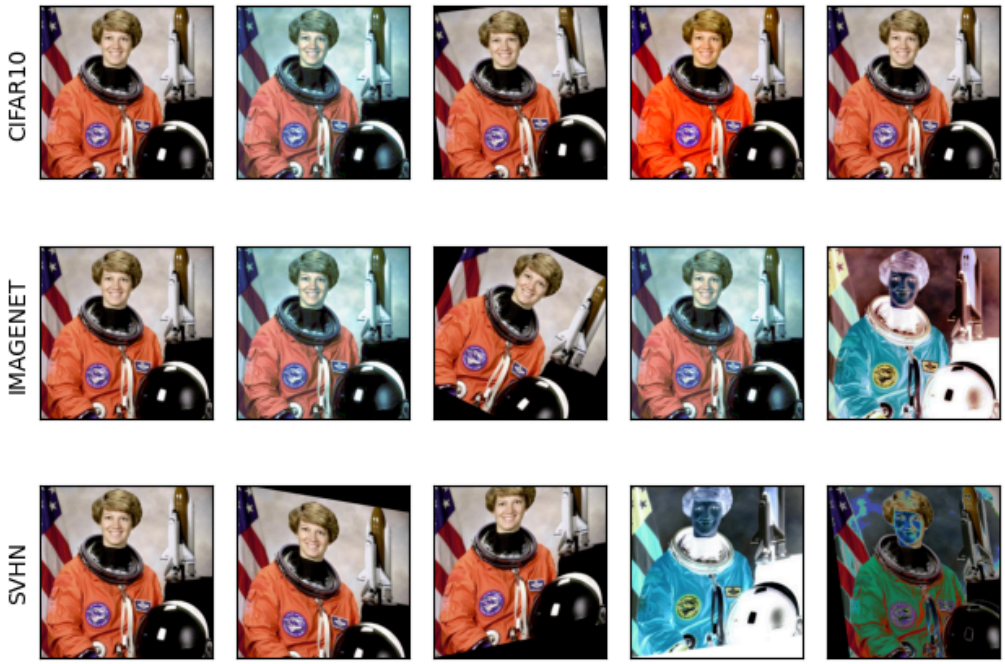
References

[1] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, “AutoAugment: Learning Augmentation Policies from Data.” arXiv, Apr. 11, 2019. Accessed: Apr. 22, 2024. [Online]. Available: http://arxiv.org/abs/1805.09501
[2] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le, “RandAugment: Practical automated data augmentation with a reduced search space.” arXiv, Nov. 13, 2019. doi: 10.48550/arXiv.1909.13719.
[3] D. Hendrycks, N. Mu, E. D. Cubuk, B. Zoph, J. Gilmer, and B. Lakshminarayanan, “AugMix: A Simple Data Processing Method to Improve Robustness and Uncertainty.” arXiv, Feb. 17, 2020. doi: 10.48550/arXiv.1912.02781.
[4] D. Rolnick, A. Veit, S. Belongie, and N. Shavit, “Deep Learning is Robust to Massive Label Noise.” arXiv, Feb. 26, 2018. Accessed: Apr. 22, 2024. [Online]. Available: http://arxiv.org/abs/1705.10694
[5] E. Weber, D. P. Papadopoulos, A. Lapedriza, F. Ofli, M. Imran, and A. Torralba, “Incidents1M: a large-scale dataset of images with natural disasters, damage, and incidents.” arXiv, Jan. 11, 2022. doi: 10.48550/arXiv.2201.04236.

[6] S. Zagoruyko and N. Komodakis, “Wide Residual Networks.” arXiv, Jun. 14, 2017. doi: 10.48550/arXiv.1605.07146.
[7] A. Dosovitskiy *et al.*, “An Image is Worth 16×16 Words: Transformers for Image Recognition at Scale.” arXiv, Jun. 03, 2021. doi: 10.48550/arXiv.2010.11929.

Appendix

I. Visualization of augmentation strategies implemented



Appendix Figure 1. *Visualisation of augmentation produced via AutoAugment class. For our project we used the IMAGENET policy. Source: https://pytorch.org/vision/stable/auto_examples/transforms/plot_transforms_illustrations.html*

Appendix Figure 2. *Visualisation of augmentation produced via RandAugment class. Source: https://pytorch.org/vision/stable/auto_examples/transforms/plot_transforms_illustrations.html*

Appendix Figure 3. *Visualisation of augmentation produced via AugMix class. Source: https://pytorch.org/vision/stable/auto_examples/transforms/plot_transforms_illustrations.html*

II. Details of the image transformations

For ResNet50
https://pytorch.org/vision/stable/models/generated/torchvision.models.resnet50.html#torchvision.models.ResNet50_Weights:
The inference transforms are available at ResNet50_Weights.IMAGENET1K_V2.transforms and perform the following preprocessing operations: Accepts PIL.Image, batched (B, C, H, W) and single (C, H, W) image torch.Tensor objects. The images are resized to resize_size=[232] using interpolation=InterpolationMode.BILINEAR, followed by a central crop of crop_size=[224]. Finally the values are first rescaled to [0.0, 1.0] and then normalized using mean=[0.485, 0.456, 0.406] and std=[0.229, 0.224, 0.225].

For Wide ResNet50
https://pytorch.org/vision/stable/models/generated/torchvision.models.wide_resnet50_2.html#torchvision.models.wide_resnet50_2:
The inference transforms are available at Wide_ResNet50_2_Weights.IMAGENET1K_V2.transforms and perform the following preprocessing operations: Accepts PIL.Image, batched (B, C, H, W) and single (C, H, W) image torch.Tensor objects. The images are resized to resize_size=[232] using interpolation=InterpolationMode.BILINEAR, followed by a central crop of crop_size=[224].

Finally the values are first rescaled to [0.0, 1.0] and then normalized using mean=[0.485, 0.456, 0.406] and std=[0.229, 0.224, 0.225].

III. Hyperparameter tuning details

loss	accuracy	time_total_s	epochs	batch_size	lr	outer_fold	inner_fold
2.444518	0.241809672	843.8070676	5	64	0.0008301	0	0
2.2489027	0.271450858	288.7463298	20	32	0.0098245	0	0
1.9608258	0.329173167	346.0731213	10	32	0.0004095	0	0
3.0762817	0.115444618	81.37569952	5	64	0.0081005	0	0
2.8933227	0.126365055	85.51971126	5	64	0.0017179	0	0
2.9056141	0.3265625	1386.689075	20	32	0.0001163	0	1
2.7622342	0.3125	1091.989386	10	32	0.0001371	0	1
2.3313208	0.2890625	294.9908519	20	32	0.0014556	0	1
2.6519645	0.1265625	90.58318257	10	64	0.0001358	0	1
2.3578856	0.2328125	283.3692286	10	32	0.000183	0	1
2.0427143	0.3578125	709.957051	10	32	0.0004536	0	2
2.5316521	0.190625	79.54036045	10	32	0.000412	0	2
2.6638365	0.140625	79.8602376	20	64	0.0011955	0	2
3.4445552	0.125	78.54218745	10	64	0.0042955	0	2
3.1664438	0.16875	77.41071701	5	64	0.0054694	0	2
2.8392631	0.391575663	1531.414467	20	32	0.000677	1	0
1.9252444	0.287051482	305.7838752	10	32	0.0080098	1	0
3.0232186	0.185647426	81.88824773	5	32	0.0004207	1	0
2.7434711	0.185647426	83.59157491	5	32	0.0009015	1	0
2.6378108	0.187207488	85.2903583	10	64	0.0041957	1	0
2.3191578	0.265625	360.8975601	5	64	0.0002182	1	1
5.4333778	0.0828125	82.70585108	5	64	0.0005782	1	1
6.5794712	0.0828125	80.26118612	10	64	0.0004712	1	1
4.5246571	0.0828125	84.12990141	5	64	0.0004697	1	1
5.2740538	0.0828125	94.63077569	10	32	0.0001022	1	1
2.0615329	0.2671875	1462.282066	20	32	0.0019155	1	2
2.9610008	0.175	85.08902073	10	64	0.0003682	1	2
6.2238181	0.16875	83.6677475	20	32	0.0034019	1	2
24.849269	0.14375	84.75033712	10	32	0.0002241	1	2
499.43816	0.0921875	83.5789578	5	32	0.0050497	1	2
3.0186381	0.308892356	1452.381942	20	32	0.0004575	2	0
3.1310667	0.146645866	81.46071458	20	32	0.0017654	2	0
2.3888375	0.291731669	294.2901559	5	32	0.0015574	2	0
2.1179472	0.319812793	293.6878314	10	32	0.0002451	2	0
2.6492192	0.204368175	81.56920052	20	64	0.0003182	2	0
2.8983859	0.32449298	1446.116662	20	64	0.0001121	2	1
2.5461906	0.23400936	294.488807	10	32	0.0017161	2	1
2.7113961	0.313572543	1120.114105	20	64	0.0009803	2	1
2.2515702	0.379095164	1407.880165	5	32	0.0082946	2	1
2.5736072	0.31201248	1156.748719	10	32	0.0079881	2	1

2.3548503	0.1734375	719.9357619	10	32	0.0093557	2	2
2.5878781	0.1015625	312.8029268	10	32	0.0026534	2	2
2.4194498	0.1484375	298.161412	5	32	0.0001389	2	2
58.895248	0.08125	89.98522592	20	64	0.0009397	2	2
2.5290637	0.13125	303.5708268	10	32	0.0001127	2	2
2.4061625	0.141965679	378.6203389	5	64	0.003233	3	0
3.2047436	0.092043682	314.6461587	20	32	0.0042586	3	0
5343.9328	0.079563183	84.97327518	10	64	0.0005638	3	0
6.1360579	0.074882995	84.95174503	5	32	0.0001787	3	0
54745.672	0.081123245	86.2695961	20	64	0.0043173	3	0
2.1245718	0.254290172	1580.006551	20	32	0.0042378	3	1
62.4919	0.095163807	112.8610082	5	64	0.0034994	3	1
385.47129	0.079563183	87.7251389	20	64	0.0009618	3	1
280.26705	0.08424337	91.69970608	10	64	0.0001737	3	1
6.3249269	0.090483619	89.31267929	5	64	0.0002391	3	1
3.0194951	0.35	1515.005401	20	32	0.000956	3	2
5.5196662	0.165625	88.03211737	10	32	0.001437	3	2
6.9704687	0.16875	86.12416887	10	32	0.0001621	3	2
7.5951521	0.15	87.3568449	5	32	0.0076165	3	2
4.5145849	0.16875	86.53140497	10	64	0.0001429	3	2
5.9894299	0.163806552	397.4357936	5	64	0.0017747	4	0
7632.5008	0.07800312	92.45949435	10	64	0.0003143	4	0
3.8019254	0.180967239	328.1034176	5	32	0.0037205	4	0
2.6754899	0.209048362	397.6359525	20	32	0.0045264	4	0
376.75204	0.096723869	94.65126777	10	64	0.0007297	4	0
2.3683489	0.15600624	802.5736468	10	32	0.0080107	4	1
2.263784	0.230889236	754.676471	10	32	0.0006037	4	1
2.4334144	0.121684867	314.2981288	20	64	0.0004105	4	1
2.2574406	0.204368175	747.2015362	5	32	0.0099676	4	1
256.51485	0.073322933	87.28542447	5	64	0.002014	4	1
2.2731561	0.203125	747.6189709	10	32	0.0098211	4	2
2.2403876	0.2171875	752.084625	20	64	0.0014507	4	2
275.9543	0.1078125	87.2135303	5	32	0.0055792	4	2
85.512407	0.084375	86.07211709	5	64	0.0001745	4	2
572.94727	0.075	86.45182133	5	64	0.0072824	4	2

Appendix Table 1. *Details of tuner experiments per each inner/outer fold for ResNet50. The configuration for the hyperparameters are given by the columns 'epoch', 'batch_size' and 'lr'. Highlighted in orange the best result for each inner fold, in red the best result overall.*

loss	accuracy	time_total_s	epochs	batch_size	lr	outer_fold	inner_fold
2.0135524	0.276131	1656.24823	10	64	0.0064518	0	0
4137258.8	0.0826833	95.87885571	10	32	0.0044789	0	0
77669.091	0.0826833	87.95341325	20	32	0.0090724	0	0
782.94273	0.0842434	94.9743824	10	32	0.0011385	0	0
9496202.9	0.0842434	90.75651836	20	64	0.0008963	0	0
2.0972832	0.290625	1547.026404	20	64	0.0012567	0	1
1661.6787	0.08125	104.9415076	10	64	0.0018391	0	1
120.80667	0.0953125	85.57029462	10	32	0.00011	0	1
14.799468	0.175	1260.638814	10	32	0.0023303	0	1
20602.591	0.0828125	97.66546297	10	64	0.0022919	0	1
2.112708	0.3046875	1535.406898	20	32	0.0019711	0	2
1078.2447	0.090625	104.2277215	5	32	0.0018973	0	2
55.359842	0.0875	111.0650065	20	32	0.0002631	0	2
2.3782521	0.196875	311.3505464	10	32	0.0022862	0	2
2.3932629	0.1953125	315.4669323	10	64	0.0056026	0	2
2.5568205	0.3416537	1743.971443	20	64	0.0011002	1	0
337.30568	0.0936037	92.93803024	5	64	0.0015416	1	0
3.1497514	0.1092044	108.2545154	5	32	0.0080218	1	0
28.53904	0.1669267	354.4877872	5	64	0.009234	1	0
3.9187697	0.1123245	92.62202048	20	64	0.003123	1	0
3.1735923	0.18125	800.6133602	10	32	0.0016363	1	1
51.796007	0.096875	100.1852372	5	64	0.0001922	1	1
132.91427	0.075	95.72831273	10	32	0.0005236	1	1
3.8885014	0.215625	332.1791203	10	32	0.0038234	1	1
2.7348844	0.1796875	346.961977	5	64	0.0061267	1	1
2.3786011	0.2109375	414.1975317	5	64	0.0013124	1	2
63.042843	0.078125	95.36060119	10	64	0.0012393	1	2
2.5122794	0.184375	333.4926341	10	64	0.0003894	1	2
9.3515099	0.11875	89.86043143	10	32	0.0002035	1	2
1099.9951	0.1109375	90.44977856	10	32	0.0005586	1	2
2.0232357	0.3463339	1605.015325	20	32	0.0009863	2	0
2.2756266	0.3088924	1340.664208	5	64	0.0001383	2	0
1.9012821	0.3712949	1690.489108	5	32	0.0001078	2	0
6.7711843	0.1981279	96.86617231	20	32	0.0033809	2	0
121.30939	0.1216849	91.16222262	20	32	0.0004577	2	0
4.1071535	0.3338534	830.7383175	10	32	0.0003114	2	1
2.4077648	0.3447738	835.887733	20	64	0.0002444	2	1
2.5697204	0.3354134	825.943568	20	64	0.0030447	2	1
2.1272348	0.3229329	364.7999735	10	64	0.0006292	2	1
3.5343335	0.1965679	111.01211	5	64	0.0005899	2	1

2.2118235	0.25625	1668.31186	20	64	0.0021651	2	2
270.41882	0.08125	112.2758126	5	32	0.0062703	2	2
14529.654	0.084375	92.3243506	5	32	0.002765	2	2
11223.97	0.071875	93.35325861	5	64	0.0070227	2	2
911.07871	0.0640625	96.14848423	20	32	0.000853	2	2
2.0755366	0.3057722	433.0884912	5	32	0.0006492	3	0
2.3042568	0.2542902	346.0400102	5	32	0.002331	3	0
3.8929089	0.1872075	116.8194368	5	64	0.0004879	3	0
7.4166118	0.1856474	98.90187597	10	32	0.0020276	3	0
2.537375	0.2464899	425.7860112	20	32	0.0051008	3	0
2.5437844	0.2043682	846.359869	10	64	0.0013637	3	1
2.5869961	0.2215289	828.9741521	5	32	0.0001991	3	1
556.37971	0.074883	89.73468828	10	64	0.0015692	3	1
22.849123	0.0795632	107.5411661	10	64	0.0007731	3	1
29.229339	0.1809672	353.0870147	5	32	0.0009066	3	1
2.3892491	0.19375	975.2274091	5	64	0.0047137	3	2
50.392595	0.0921875	361.3553207	10	32	0.0002077	3	2
18573917	0.0828125	90.67012429	20	32	0.000525	3	2
6258.5787	0.084375	100.3416924	5	64	0.0074136	3	2
63118.069	0.08125	97.20786381	5	32	0.0018979	3	2
3.1212864	0.2948518	821.0232575	10	64	0.0003041	4	0
4.9128968	0.1029641	98.67080379	20	32	0.0018874	4	0
3.821334	0.2917317	898.4393392	20	32	0.0020307	4	0
4.2757944	0.1138846	98.95956898	10	32	0.0031789	4	0
5.0028617	0.1107644	97.13151622	10	32	0.0018508	4	0
2.0225164	0.299532	1718.978152	20	32	0.001221	4	1
25.716881	0.1778471	104.0309579	10	32	0.0017883	4	1
106.75682	0.1279251	103.4644015	10	32	0.0007046	4	1
4.0323199	0.1950078	98.97576451	10	32	0.0002457	4	1
18.979028	0.1544462	109.0079374	10	64	0.000383	4	1
103907605	0.078125	104.2991245	20	32	0.0053405	4	2
3.155307	0.1	333.998461	5	32	0.0048635	4	2
158993759	0.0828125	114.291847	10	64	0.0001042	4	2
16676134	0.0828125	100.0387397	5	64	0.0029148	4	2
12.470464	0.1546875	596.311049	5	64	0.0089389	4	2

Appendix Table 2. *Details of tuner experiments per each inner/outer fold for Wide ResNet50. The configuration for the hyperparameters are given by the columns 'epoch', 'batch_size' and 'lr'. Highlighted in orange the best result for each inner fold, in red the best result overall.*

All available tunable parameters offered by our code implementation:

- "epochs": number of epochs to run training

- "batch_size": the batch size
- "lr": the learning rate for the optimizer
- "weight_decay": weight decay parameter for some optimizers like AdamW (default)
- "scheduler_factor": parameter for the scheduler
- "scheduler_patience": patience for the scheduler
- "criterion": (loss function in torch.nn) a valid loss function, default to CrossEntropy
- "optimizer": (valid optimizer object in torch.optim)
- "scheduler": (valid scheduler object in torch.optim)

IV. Optimization and code development

In our project, we significantly enhanced our code development process by integrating Ray, a powerful library designed for parallel and distributed computing. This strategic adoption allowed us to handle computationally intensive tasks more efficiently, especially critical as we scaled our data processing and machine learning model training phases. Although we weren't able to leverage the full power Ray offers in distributing efficiently workloads in cluster nodes, since we had access to single machines only, our code already accommodates the possibility of scaling up and transfer the processes in clusters with more available resources.

All our code was carefully crafted into a Python package that was used to obtain all the results in this project.