Evaluation of generalisation accuracy on Deep features from ImageNet

CSCI946 Big data Analytics

Assignment 3

University of Wollongong

Group 8

Group members

Full Name	Student ID	Responsibility	Contribution (%)
Robert Mete	5736262	Perform advanced analysis and visualisation	100
Kyuta Yasuda	6588773	Literature review for model selection	100
Mikael Shahly	8945512	Perform advanced analysis and visualisation	100
Thamonwan Nitatwichit	8026300	Build the model with the best classification and write data analytics lifecycle	100
Huu Thien Pham	7794587	Data preprocessing and code refactoring	100
Syed Eisa Alamgir	8188154	Perform visualisation	70
Anas Shahid Raja	8279366	Perform visualisation	70

Subject coordinator

Professor Lei Wang

Table of content

Background

The rapid growth of digital images and videos has made visual data analysis a core aspect of **Big Data Analytics**, requiring advanced algorithms for efficient processing. Deep learning models play a crucial role in extracting complex patterns from images, driving applications such as **surveillance, autonomous vehicles, healthcare diagnostics, sentiment analysis, and security systems**. This integration demands specialised expertise, robust computational resources, and **hardware like GPUs or TPUs** to handle large datasets effectively.

Pre-trained models have become increasingly popular as feature extractors, enabling faster development of solutions by minimising retraining efforts. **ImageNet**, introduced in 2009, has been a benchmark for image recognition, propelling advancements with competitive challenges. However, reliance on the original ImageNet validation set (referred to as **Test Set 1**) has raised concerns about **overfitting**, as models risk becoming too specialised to its 50,000 images. To address this, **ImageNetV2** introduced new validation sets like **MatchedFrequency**, referred to as **Test Set 2**, offering a similar class distribution to evaluate generalisation more effectively.

While differences in performance across these test sets persist, **Recht et al. (2019)** found that high-performing models on Test Set 1 also perform well on Test Set 2, suggesting that **image difficulty and dataset composition**, rather than overfitting, drive performance variability. These findings highlight the need for more diverse benchmarks to accurately measure model generalisation across datasets.

Big data life cycle from theory to Project Design

Phase 1: Discovery

Frame the problem

Regarding the background of our project, the goal is to prove whether models trained on the ImageNet dataset (Test Set 1) generalise effectively to the ImageNetV2 dataset (Test Set 2). We hypothesise that performance discrepancies arise due to subtle differences in the datasets, such as image difficulty, rather than overfitting. This aligns with the findings of Recht et al. (2019), who suggest that while accuracy drops are observed between original and new datasets, they are driven more by dataset complexity than model adaptivity (Recht et al., 2019).

Identify key stakeholders

Key stakeholders will include course instructors, project team members, and research community interested in model generalisation and computer vision benchmarks.

Develop initial hypotheses

We hypothesise that model performance drops between Test Set 1 and Test Set 2 are not caused by overfitting but by complex images and noise in images in Test Set 2, consistent with Recht et al.'s (2019) analysis.

Identify potential data sources

Our project has been supplied with guidelines, related files, and ImageNet dataset (Test Set 1) and ImageNetV2 dataset (Test Set 2), importantly the dataset of extracted features from the pre-trained model.

Phase 2: Data preparation

Explore and pre-process data

Extract the contents of the provided datasets and convert data into a consistent format suitable for analysis (e.g., CSV files for features extracted from pre-trained models). If further preparation regarding the size of the dataset is needed, the dataset will be quantised for computational practice.

Prepare analytics sandbox

Set up the development environment using Python and PyTorch with necessary libraries and dependencies (e.g., Hugging Face library).

Perform ELT (Extract, Transform, Load)

Prior to our project, the pre-trained models were used to extract deep features from images in Test Set 1 and Test Set 2, after that these features were transformed into numerical datasets, ensuring consistent feature dimensions, quantising the data set to reduce its size for computational efficiency and ready for further steps.

Data conditioning

Clean and validate the extracted features, ensuring they are suitable to be the input of the chosen models.

Phase 3: Model planning

Identify candidate models

Nowadays, there are several machine learning models and different architectures of neural networks for us to address the problems. Basically, the problem nature falls in classification domain, thus, we list all potential classifier models that fit the task as below.

Logistic regression: It is a statistical and machine learning method used to predict the probability of a binary outcome (e.g., yes/no, success/failure). It belongs to the family of supervised learning models and is often applied in scenarios where the dependent variable has two classes (binary logistic regression), but it can also handle multiclass problems (multinomial and ordinal logistic regression) (Analytics Vidhya, 2024; IBM, n.d.).
Random Forest: It is an ensemble machine learning algorithm that builds multiple decision trees and aggregates their predictions to enhance model accuracy and reduce the risk of overfitting. It is versatile and can be applied to both classification and regression tasks, making it a popular tool in machine learning (IBM, n.d.; Analytics Vidhya, 2024).
CatBoost: It was developed by Yandex, is an open-source gradient boosting algorithm designed to handle both numerical and categorical features efficiently. It is particularly well-suited for tasks such as classification, regression, and ranking, and is used in diverse applications, including recommendation systems, fraud detection, and natural language processing (CatBoost, n.d.; Built In, 2024).
eXtreme Gradient Boosting (XGBoost): It is a high-performance, open-source machine learning library that belongs to the ensemble learning family. It builds an ensemble of decision trees, where each new model focuses on correcting the errors made by previous ones. This iterative improvement process is based on gradient boosting, a technique that optimises predictions using gradient descent to minimise a loss function (Analytics Vidhya, 2024; IBM, n.d.; Machine Learning Models, 2024).
Feed-Forward Neural Network (FFN): It is a simple neural network where information flows in one direction—from the input layer, through hidden layers, to the output layer—without feedback loops, unlike recurrent neural networks (RNNs). The input layer receives data, such as images, text, or numerical features, which is processed through hidden layers that apply linear transformations and activation functions (like ReLU or Sigmoid) to capture patterns. The output layer generates predictions, such as class labels in classification tasks or continuous values in regression tasks. During forward propagation, each neuron computes a weighted sum of inputs, applies an activation function, and passes the result to the next layer. Predictions are evaluated against actual values using a loss function (e.g., Cross-Entropy or Mean Squared Error). In backpropagation, errors are propagated backward, adjusting weights with optimization algorithms like Stochastic Gradient Descent (SGD) or Adam. FFNs are

commonly used for classification tasks (e.g., image recognition, sentiment analysis), regression tasks (e.g., predicting prices of stock trends), and function approximation.

Their simplicity makes them easy to implement and effective for structured data. FFNs are foundational models in deep learning and are often combined with other architectures, such as convolutional or recurrent layers, to address more sophisticated tasks.

Literature review

Review relevant research and documentation on the performance, advantages, and limitations of each candidate model. Focus on their applications in similar tasks, such as generalisation performance on new datasets, to align with the project goals.

Data exploration and variable selection

Conduct exploratory data analysis (EDA) to identify the most relevant features for model training, including visualising distributions and correlations to detect patterns, handling missing values and outliers, and reducing noise by removing irrelevant variables.

Model selection

CatBoost and Feedforward Neural Networks (FFNs) are strong candidates for image classification due to their complementary strengths. **CatBoost** is particularly advantageous for handling structured data with categorical features, as it eliminates the need for preprocessing (e.g., one-hot or label encoding), saving time and computational resources. While traditionally used for tabular data, CatBoost's ability to efficiently manage categorical inputs can be leveraged if the dataset includes metadata or features beyond raw images (e.g., labels, tags, or descriptive attributes).

On the other hand, **FFNs** are well-suited for learning patterns within numerical data, such as pixel intensities from flattened images or embeddings derived from deep models. FFNs perform well when the dataset is not excessively complex, as they are easy to implement, computationally efficient, and effective at mapping inputs to outputs through non-linear transformations. Additionally, their structure allows for quick experimentation with various hyperparameters and architectures, making them ideal for limited-resource environments.

By combining **CatBoost** and **FFNs**, we cover different aspects of the image classification task: FFNs excel at learning from the numerical image data, while CatBoost can handle any structured, categorical metadata that might enhance the prediction accuracy. This approach ensures a balance between performance and efficiency, with CatBoost offering regularisation to prevent overfitting and FFNs providing a flexible architecture for fast convergence. Together, these models offer a robust framework for experimentation, enabling efficient training, fast inference, and reliable generalisation across unseen data.

Phase 4: Model Building

Once we finalise which models might suit our task the most and be computationally efficient, we will continue from the transformed data from phase 3. Dimensional reduction techniques will be used before feeding the data to our models, ensuring that redundant features will be removed from all 1024 features. Next, the prepared data will be split into a training set for model training and validation set for hyperparameter tuning step.
Use both test set 1 and test set 2 for final performance evaluation. Implement models using scikit-learn or PyTorch. After that, team members will investigate deep features and analyse the factors or features that cause a gap in accuracy between both test sets. If the factors are spotted, the analysis team will make adjustments and retrain or retest the models with adjusted data for hypothesis justifications. For advanced analysis, there will be sections apart from model building for elaboration.

Tools: Python, PyTorch, scikit-learn, Hugging Face libraries

Phase 5: Communicate the results

Outcome assessment

Report performance metrics (e.g., accuracy, precision, recall) for Test Set 1 and Test Set 2. Compare these results to report the model's generalisation ability.

Communicate with members and stakeholders

Hold regular meetings with team members to discuss progress and challenges. Maintain clear documentation of tasks and responsibilities.

Recommendations for future work

1. Investigate the use of data augmentation techniques to improve generalisation.
2. Explore additional classifiers for improved generalisation accuracy.
3. Adjustments in architecture of the model
4. Conduct further analysis to understand which feature sets work well across different test sets.

Phase 6: Operationalise

Potentially, the models can be integrated with systems related to the **ImageNet dataset** or adjustments can be proposed to fine-tune the models for deployment, ensuring the models are not only trained but also be able to be **deployed in real-world systems** to deliver continuous insights or predictions. For example, the CatBoost model could be integrated to handle **structured metadata** accompanying image data, such as labels or tags, while the FFN processes raw **image embeddings** or pixel values. Adjustments to the ImageNet dataset, such as **re-sampling, re-labeling, or fine-tuning with specific image subsets**, can improve model relevance and ensure optimal performance in the target environment.

Further steps include **monitoring the deployed models** to track performance, retraining with new data to maintain accuracy, and ensuring the system can **scale efficiently** with increased data volume. By operationalizing the models in this way, we can ensure smooth integration into existing workflows, allowing them to deliver meaningful and actionable predictions in real-time applications.

Discovery

Problem definition

From the past to present, little research was conducted to determine the features and factors that hinder the consistency of model performance on different two test sets of ImageNet. Therefore this project is focused on improving the generalisation ability of an image classification model on the ImageNet test set and the ImageNetV2 test set. The project's main goal is understanding the significant classification performance gap of models on the ImageNet validation set (frequently referred to as validation set 1) and the matched frequency split of the validation set ImageNetV2 (referred to as validation set 2). Based on this understanding, another goal is then find concrete recommendations to reduce this performance gap.

Stakeholders

As the availability of visual data, and our dependence on being able to correctly interpret it, increases it becomes vital to develop models that perform well not only in controlled environments, but also in real world scenarios. This project's importance spans multiple industries that heavily rely on visual data to perform decision making. As such, important stakeholders could be the healthcare industry (used for diagnostics), security (for surveillance) and automobile industry (for self driving autonomous vehicles).

Developing initial hypotheses

We hypothesise that performance discrepancies arise due to subtle differences in the datasets, such as image difficulty, rather than overfitting. This aligns with the findings of Recht et al. (2019), who suggest that while accuracy drops are observed between original and new datasets, they are driven more by dataset complexity than model adaptivity (Recht et al., 2019).

Data Preparation

Data Extraction and Overview

Data preparation processes start with extraction. The initial datasets are provided through a cloud storage platform that is hosted by the University of Wollongong. These datasets are in the form of compressed CSV (comma-separated values) files which add up to just over 7 gigabytes. After the data is carefully examined, it comes to light that the data needs to be uploaded to another platform that can facilitate machine learning and group collaboration.

Kaggle is a reasonable choice, offering more than sufficient storage for 15 gigabytes of uncompressed data and enough computing power for preliminary pre-processing, which is discussed below. After being uploaded, the data files are decompressed and made ready for further transformations. Google Colab is another option, but in the attempts of exploring and transforming the datasets there, memory issues consistently occur because the whole training set cannot be loaded into the RAM (random-access memory) of Colab's Jupyter notebooks, which only hold just below 13 gigabytes of data.

In this project, a training set and two test sets are provided separately, so there is no need for such splitting. There are close to 1.3 million records in the training set, 50 thousand in test set 1 and 10 thousand in test set 2. The target features in these sets are made up of 1000 unique labels and are considerably balanced. On the other hand, every set has 1024 predicting float-type features, extracted from a hidden layer of a pre-trained image recognition model.

Data Transformation (Preliminary Processing)

The primary preprocessing was conducted to prepare the dataset for classification. First, we applied **standard scaling** to normalise each feature's values, ensuring they all operated on the same scale. This normalisation process is crucial; without it, features with larger numerical values could disproportionately influence model training, skewing the results. By bringing all features to a common scale, we improve the performance of machine learning algorithms sensitive to data scaling, such as Multilayer Perceptrons (MLP), which is one of our candidate models.

An important note for the standard scaler's application is that it should first be fit to the training set only, and then applied to transform both the training and validation sets. If the scaler is applied across the entire dataset before splitting it into training and validation sets, it risks **data leakage**, as it would use information from the validation set (through min and max values across the entire dataset) to scale the training data, inadvertently influencing the training process with future validation data. This data leakage could compromise the validity of our experiment, leading to an inflated assessment of model performance.

Model Planning

To tackle the challenges of our big data analytics task, we need to address the dataset's substantial size, with over 1 million records and more than 1,000 features—each feature representing model-derived weights, resulting in high-dimensional data. This volume and complexity demand machine learning models that are not only computationally efficient but also capable of handling large-scale, high-dimensional data while maintaining strong classification performance. Additionally, the selected models must be able to leverage parallel computing to optimise processing times and resource use.

Given that the dataset is structured, of high quality, and comprises entirely numerical features, we have identified two models that are particularly well-suited for this task: an **e**nsemble boosting model—specifically CatBoost—and a Multi-Layer Perceptron (MLP) implemented using TensorFlow/Keras.

CatBoost is an ensemble boosting algorithm that excels in handling large datasets with high dimensionality. It is designed to process vast amounts of data efficiently by utilising optimised algorithms that reduce memory consumption and computation time. CatBoost inherently supports parallel computing, allowing it to leverage multi-core processors and GPU acceleration—crucial features for managing the computational demands of big data.

The algorithm combines multiple weak learners to form a strong classifier, enhancing overall prediction accuracy. It effectively handles numerical features and can automatically process categorical features if present. CatBoost also incorporates techniques like ordered boosting and minimal variance sampling to reduce overfitting, thereby improving the model's generalisation capabilities.

The MLP, implemented via TensorFlow/Keras, offers the flexibility and modelling power necessary to capture complex patterns within the data. Neural networks like MLPs are adept at modelling non-linear relationships, which is particularly beneficial when working with features derived from deep learning models. TensorFlow/Keras facilitates efficient training on large datasets through support for GPU acceleration and distributed computing, enabling the handling of big data more effectively.

Customising the MLP's architecture—including the number of layers, neurons, and activation functions—allows us to balance computational efficiency with model performance. This adaptability is essential for experimenting with different configurations to identify the optimal setup for our classification task. Furthermore, TensorFlow/Keras provides robust tools and a supportive community, which enhances development and troubleshooting processes.

Selecting CatBoost and the MLP addresses the critical need for models capable of efficiently handling large-scale, high-dimensional data while maintaining robust classification performance. CatBoost's strengths lie in its ability to manage structured numerical data and exploit parallel computing resources effectively. Its ensemble nature contributes to high prediction accuracy and robustness against overfitting.

The MLP complements CatBoost by offering the ability to model intricate non-linear relationships inherent in the deep learning features. The use of TensorFlow/Keras enhances the MLP's capabilities through efficient computation and flexibility in model design. This combination allows us to explore different algorithmic approaches, providing a comprehensive analysis and comparison to select the model that offers the best performance for our classification task.

By employing these models, we aim to achieve a balance between computational feasibility and high classification accuracy. Both models are expected to effectively handle the scale and complexity of our dataset, facilitating the development of a robust classifier. This strategic model planning is integral to advancing through the big data analytics lifecycle, ultimately working towards reducing the performance gap between the ImageNet and ImageNetV2 test sets and improving overall model generalisation.

Model Building

Preparing the train set before the models

Dimensional reduction

Principal Component Analysis (PCA) was used to reduce the dataset's dimensionality from 1,024 to 643 features, retaining 90% of the original variance. Similar to how not all pixels equally contribute to an image, not all deep learning features hold the same significance. PCA removes redundant features, improving computational efficiency, reducing memory usage, and enabling faster experimentation—critical given our hardware constraints. It also mitigates the curse of dimensionality, enhancing model performance.

Although PCA makes features less interpretable, this is not a concern since the original features are abstract weights from a pre-trained model, already difficult to interpret. Ultimately, PCA strikes a balance between dimensionality reduction and information retention, preserving essential data and ensuring high classification accuracy. Research further supports that eliminating noise and redundancy through dimensionality reduction can enhance model performance.

These preprocessing steps, applied uniformly across all four datasets, ensure consistency in data transformation throughout the process. This approach significantly reduces computational load and training time, enabling us to build a more computationally efficient classifier. Given the abstract nature of the original features and its large size, the slight loss in interpretability is an acceptable compromise. This preprocessing strategy allowed us to focus on enhancing the model's performance and narrowing the performance gap between the ImageNet and ImageNetV2 test sets.

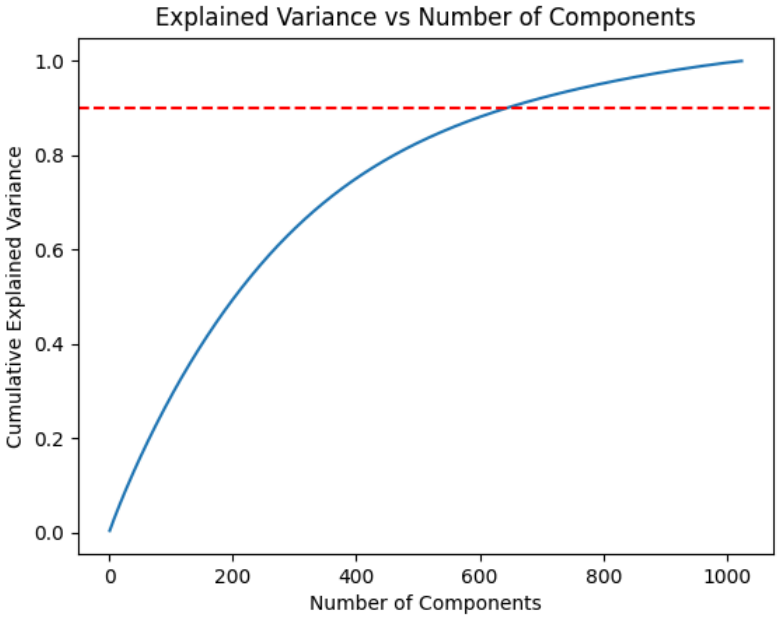


Figure 1: The plot representing the relationship between cumulative explained variance and number of components

Splitting dimension-reduced train set

The original training set is exceptionally large, comprising 1,281,168 records that span across 1,000 different class labels. Each record includes 1,024 features, with each feature representing the output weight from a layer of a pre-trained computer vision model. Managing such a voluminous dataset demands substantial computational resources and leads to extensive training times and hyperparameter tuning times, which did not align with our computational limitations.

To address these constraints for all candidate models, we employed **stratified sampling** to reduce the size of the dataset while preserving the original class distribution. Stratified sampling ensures that the proportion of each class in the sampled dataset mirrors that of the full dataset. This approach prevents class imbalance, which could adversely affect the model's performance by introducing bias toward overrepresented classes. By maintaining the class distribution, we mitigate the risk of bias in the training process and enhance the reliability of the classification results on the reduced dataset. Therefore, the whole train set (1,281,168 records) was split into 80% for the training set and the remaining 20% for the validation set to be fed into all models. However, each candidate utilised the available computational resources differently based on their distinct nature, leading to excessive will be elaborated further in their sections.

To all candidate models, it is important to note that **no subsampling was applied to either of the test sets**, and they were left untouched, ensuring that the evaluation of the model's performance accurately reflects its ability to generalise to new, unseen data. Altering the test

sets could introduce biases or artefacts that might skew the results and compromise the validity of our performance assessment.

By carefully managing the dataset through stratified sampling and mindful splitting, we optimised the training process to be both computationally feasible and methodologically sound. This strategy allowed us to train the classifier effectively within our resource constraints while mitigating potential biases. It ensured that our findings are robust and that the model's performance is a true reflection of its capabilities across all classes.

Working environment

Google Colab Pro was chosen as it can offer several advantages for deep learning model development, including access to high-performance GPUs and TPUs (e.g., NVIDIA T4, P100) that accelerate training, extended runtimes (up to 24 hours), and increased memory to minimise interruptions. It supports mixed precision training for enhanced efficiency and integrates seamlessly with Google Drive and GitHub for collaboration and version control. Additionally, Colab Pro provides an affordable, scalable alternative to setting up personal hardware, making it well-suited for complex tasks such as training models on large datasets like ImageNet (DataCamp, 2024; McCormick, 2024). Even so, it came with the limited number of computing units affected by the active runtime in the current session, preventing us from proceeding and completing some steps in the CATBoost model.

Hyperparameter tuning

Optuna

Optuna is an advanced hyperparameter optimization framework that offers a **"define-by-run" approach**, allowing for dynamic construction of the search space during the optimization process. One of its main strengths is its **efficiency through pruning**, where less promising trials are stopped early, saving computation time. This makes Optuna especially useful for complex models where evaluating every parameter set can be computationally expensive. Additionally, Optuna supports **distributed training** across multiple machines and provides **visualisations** to monitor optimization progress, making it user-friendly and powerful for large-scale experiments (Hashnode, 2024; Kaggle, 2024).

Random Search

Random Search, on the other hand, offers a simpler, more flexible approach by **randomly sampling from the hyperparameter space**. It is generally faster than exhaustive search methods like grid search because it does not try every combination of hyperparameters. Instead, it explores a wider variety of the search space with fewer trials, which makes it effective for **high-dimensional problems**. While Random Search may miss some optimal parameter configurations, its simplicity and efficiency make it a popular choice when **computational resources are limited** (Keylabs, 2024; Kaggle, 2024).

Both techniques offer valuable benefits: **Optuna** provides sophisticated control and scalability, while **Random Search** is straightforward and adaptable to various tasks. The choice between them depends on the complexity of the model, the availability of computational resources, and the level of optimization required. We employed Optuna with our first model (CATBoost) to first investigate the computer efficiency of Optuna with CATBoost model, however, **Optuna** demonstrated the need of high computer resource and a fairly large number of computer unit (more than 30 out of 100), leading us to make a decision to use **Random Search** for FFN instead since **Random Search** dominates Optuna in terms of less computational cost.

CATBoost classifier

Data processing

For the CatBoost classifier, the size of the training set will be optimised according to the limited computing resources, as the nature of **Gradient Boosting Models (GMM)** involves sequentially building multiple weak learners (decision trees) that can become computationally intensive. This process requires balancing the number of iterations and depth of trees to avoid overfitting while keeping memory usage and runtime manageable in the Colab pro environment. Since CatBoost performs **ordered boosting** to prevent target leakage and uses **symmetric trees** to speed up inference, optimising the dataset size is essential to maintain efficient training within resource constraints. We retained 50% of total records by sampling the train set after splitting with **0.8/0.2 ratio.** Thus, there were **approximately 50** examples per class. To ensure that both sets maintained the stratified class distribution before feeding them into the model.

Model training

Table 1: Baseline Model Evaluation on validation set

	precision	recall	f1-score	support
accuracy			0.91	256234
Macro avg	0.92	0.91	0.91	256234
Weight avg	0.92	0.91	0.91	256234

The model then was trained on a proportioned train set, yet was evaluated on 20% of the whole train set after PCA. The model exhibits strong and balanced performance across all key metrics, with an overall accuracy of 91%. Both the macro average and weighted average F1-scores (0.91) confirm that the model generalises well across different classes in the validation set. This suggests that the model performs consistently, with minimal performance variation between classes, indicating no major class imbalance or bias issues. Given the high scores for precision, recall, and F1, this model is well-suited for practical deployment. The

performance metrics indicate that the model can make reliable predictions with a low risk of both false positives and false negatives, which is critical for many applications. Further steps could focus on ensuring the model maintains this level of performance under both test sets.

CATBoost performance result

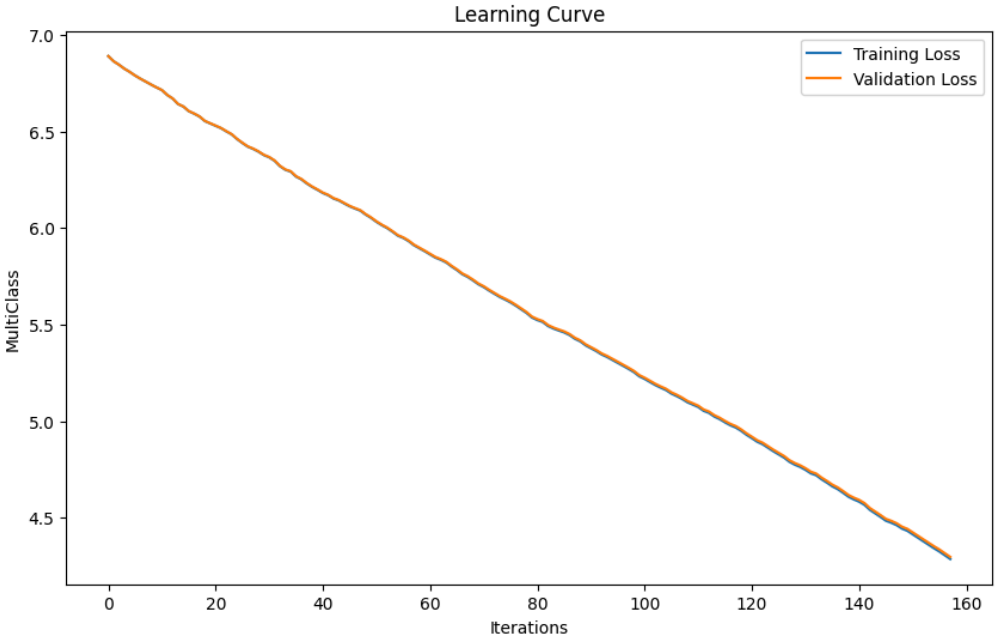


Figure 2: learning curve of CATBoost model after hyperparameter tuning

From the **figure 2**, Both the training loss (blue) and validation loss (orange) steadily decrease over the course of the training process, suggesting that the model is improving and learning meaningful patterns with each iteration. The two lines are closely aligned and decrease at a similar rate implies that the model is neither overfitting nor underfitting. If overfitting were present, the validation loss would diverge from the training loss after a certain point, increasing while the training loss continues to decrease. Furthermore, the smooth decline in both curves indicates stable learning, with no abrupt changes suggesting optimization issues. This implies that the learning rate and other hyperparameters are likely well-tuned, ensuring the model converges properly. The plot likely indicated that the model is training effectively, with the validation loss following the training loss closely, suggesting that the model generalises well to the validation data without significant overfitting. However, it is important to note that the model has been trained and evaluated from only 50% of the train set after PCA processing, this learning curve might not reflect how CATBoost would be learning through the whole train set. Therefore, our recommendation for future work is to invest more in the candidate environment to achieve higher RAM and GPU in order to investigate how CATBoost would learn across the whole train set.

Table 2A: Final Model Evaluation on test set 1

	precision	recall	f1-score	support
accuracy			0.86	50000
Macro avg	0.87	0.86	0.86	50000
Weight avg	0.87	0.86	0.86	50000

Table 2B: Final Model Evaluation on test set 2

	precision	recall	f1-score	support
accuracy			0.78	10000
Macro avg	0.79	0.78	0.77	10000
Weight avg	0.79	0.78	0.77	10000

Due to high computational cost and regard of time efficiency, we decided to preserve the remaining computer units on Colab Pro for FFN creation. Therefore, we did not finish hyperparameter tuning properly for the CATBoost classifier, instead we picked up one of the best hyperparameters studied by Optuna. Then, we simply evaluated the model directly on both test sets for initial observation and confirmation of the gap in accuracy between two test sets. In other words, there was no retraining of the model with the best parameter and evaluation on the validation set that could be split from the train set.

From table 2A and table 2B, the model performs better on Test Set 1 with an accuracy of 86%, compared to 78% on Test Set 2, indicating stronger predictive power and generalisation on the first set. The macro and weighted averages for precision, recall, and F1-score are also higher on Test Set 1 (0.87, 0.86, and 0.86) than on Test Set 2 (0.79, 0.78, and 0.77), suggesting that the model maintains a better balance between precision and recall with the larger dataset. The smaller sample size of Test Set 2 (10,000 samples) may contribute to higher variance and less stable performance compared to the 50,000 samples in Test Set 1. This disparity could indicate that Test Set 2 contains more challenging or less representative data. Further analysis could explore differences in feature distributions between the sets and consider fine-tuning or data augmentation to improve the model's robustness on smaller or more diverse datasets.

Feed-forward neural networks (FFNs)

Data processing

TensorFlow efficiently utilises GPUs for deep learning by leveraging CUDA and cuDNN libraries, enabling parallel processing across multiple cores to handle large workloads. This makes it well-suited for datasets like **ImageNet**, where feature-rich embeddings require fast and scalable computations. TensorFlow supports advanced techniques such as **mixed precision training**, which increases computational throughput by using lower-precision data types (e.g., float16) while maintaining accuracy. Moreover, TensorFlow's distributed training capabilities allow it to scale across multiple GPUs, optimising performance for large datasets (Scaler, 2024; MyScale, 2024). With all mentioned strengths, FFN was able to process the whole train set without out-of-memory suffering.

In contrast, **CatBoost** specialises in structured data with categorical features, focusing on tree-based models. Although it supports GPU acceleration, CatBoost is less effective in managing high-dimensional numerical data, making TensorFlow a more suitable option for tasks involving feature extraction from large datasets like ImageNet (Neptune.ai, 2024). This was the superior choice when the computing units in colab pro were limited to the runtime in the notebook.

Model architecture

General architecture of FFNs

A Feedforward Neural Networks (FFNs), also called a Multilayer Perceptron (MLP), is a simple neural network architecture where data flows sequentially from the input layer, through one or more hidden layers, to the output layer, with no feedback loops. Each hidden layer applies a linear transformation followed by a non-linear activation function (e.g., ReLU or Sigmoid), allowing the network to learn complex patterns. FFNs are trained using backpropagation, where the error is propagated backward to adjust weights and minimise the loss function. They are widely used for tasks such as classification and regression, especially when working with structured data (Cudo Compute, 2024; Analytics Vidhya, 2024).

The number of hidden layers in a Feedforward Neural Networks (FFNs) varies depending on the task complexity: **Shallow networks** with **one hidden layer** are often enough for simpler tasks like basic classification or regression, meanwhile for **Deeper networks**, **3–10 hidden layers** are used for more complex problems such as image and speech recognition (Analytics Vidhya, 2024). For most real-world applications, **1-3 hidden layers** offer a good trade-off between performance and computational cost. However, more layers increase the risk of overfitting, requiring more data and careful tuning (Cudo Compute, 2024)

Our proposed architectures

We prioritised the simplicity and computational efficiency before the complexity of the architecture. This led to 2 main architectures in baseline FFN model and Final FFN model.

Baseline FFN architecture

Baseline FFN consists of two hidden layers (32 and 16 neurons) and an **output layer of 1000 neurons using softmax**, the model aligns perfectly with ImageNet's **1000-class classification task**. This **architecture** was well-suited for working with **extracted features from ImageNet** because it leverages a simple, fully connected structure ideal for further learning without convolutional layers. This lightweight design strikes a balance between capturing essential patterns and avoiding overfitting, especially since the input features are already informative. The **Adam optimizer** provides efficient training with adaptive learning rates, and **sparse categorical cross entropy** is appropriate for handling class indices directly. Starting with this minimal architecture allows for performance monitoring, providing a solid baseline to build upon if further improvements or complexity are needed (Analytics Vidhya, 2024)

Final FFN architecture

It was built on the baseline but incorporating **dropout layers** to improve generalisation and prevent overfitting, especially when working with **high-dimensional features** from ImageNet. The model starts with a **Dense layer** of 32 neurons (with ReLU activation) and introduces a **dropout rate of 0.2**, randomly deactivating neurons during training to avoid reliance on specific paths. A second Dense layer with half the neurons (16) further refines the feature learning, followed by another dropout layer. The **output layer with 1000 neurons and softmax activation** matches the **1000-class structure of ImageNet**, outputting class probabilities. The **Adam optimizer** is configured with a learning rate of 0.001, balancing training speed and convergence, while **sparse categorical cross entropy** handles class indices directly. This architecture offers flexibility through hyperparameters like **learning rate, neuron count, and dropout rate**, making it robust for experimentation and fine-tuning to optimise performance (Cudo Compute, 2024; Analytics Vidhya, 2024)

Custom FFN architecture

Its architecture was built in the same way with **final FFN architecture** but the parameters were the same with baseline FFN architecture to further investigate if hyperparameter tuning by **Random Search** truly led us to the best classification performance on both test sets.

Model training

Table 3: baseline FFN Evaluation on validation set

	precision	recall	f1-score	support
accuracy			0.91	256234
Macro avg	0.92	0.91	0.91	256234
Weight avg	0.92	0.91	0.91	256234

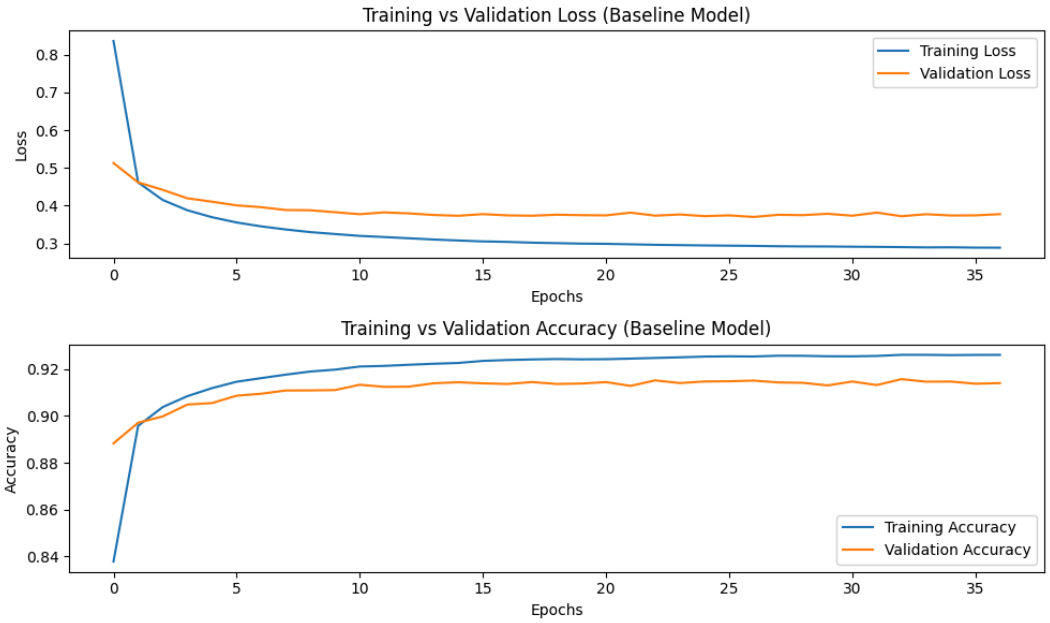


Figure 3: The learning curve plot of Baseline FFN on validation set

The table 3 presents the performance metrics for the baseline Feedforward Neural Network (FFN) on the validation set, evaluated on a total of **256,234 samples**. The model achieves an **accuracy of 91%**, meaning 91% of the samples were correctly classified. The **macro average** for precision, recall, and F1-score is **0.92, 0.91, and 0.91**, respectively, treating all classes equally without weighting based on their frequency. Similarly, the **weighted average** metrics—also 0.92, 0.91, and 0.91—confirm a balanced performance across classes, indicating no significant class imbalance. Overall, the table reflects that the model performs consistently well, with high precision and recall, suggesting effective generalisation to the validation data.

According to **figure 3**, the first sub plot shows the **training and validation loss** over 35 epochs, with both losses decreasing sharply during the initial epochs and stabilising around **epoch 10**. After this point, the validation loss plateaus, indicating that the model has likely **converged** without overfitting. The close alignment between the two loss curves further confirms that the model generalises fairly well to unseen data and avoids overfitting. Meanwhile, the second sub plot tracks the **training and validation accuracy** across the same 35 epochs. Both accuracies improve quickly during the early stages, surpassing **90% by epoch 10**. Throughout the training process, the training and validation accuracies remain closely aligned, indicating consistent performance and good generalisation. The absence of significant divergence between the two curves suggests the model is neither aggressively **overfitting** nor **underfitting**, reinforcing that the model is well-trained and considerably stable across the dataset.

Table 4: Final FFN Evaluation on validation set

	precision	recall	f1-score	support
accuracy			0.76	256234
Macro avg	0.79	0.76	0.75	256234
Weight avg	0.79	0.76	0.75	256234

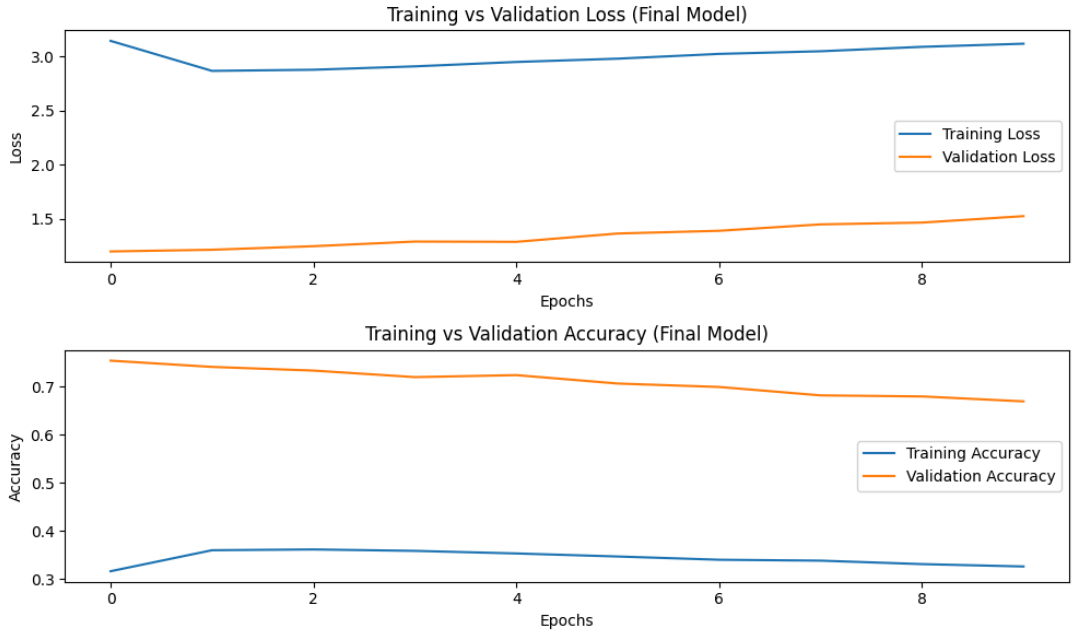


Figure 4: The learning curve plot of Final FFN on validation set

The final version of FFN model (after hyperparameter tuning) could be trained with the whole datasets using drastically less time than the CATBoost model. We could use RandomSearch to identify the best parameters: {'num_neurons': 32, 'learning_rate': 0.01, 'epochs': 10, 'dropout_rate': 0.2, 'batch_size': 128}. Then we trained the Final model with the obtained best parameters to compare the ability to learn during training of the final FFN model and baseline FFN model.

Table 4 presents the **final FFN evaluation metrics** on the validation set with **256,234 samples**. The model achieves an **accuracy of 76%**, meaning it correctly classifies 76% of the samples. The **macro average** for precision, recall, and F1-score is **0.79, 0.76, and 0.75**, respectively, treating all classes equally without regard to class size. Similarly, the **weighted average** for these metrics remains at 0.79, 0.76, and 0.75, suggesting a consistent, though moderate, performance across all classes.

Regarding the **figure 4**, the **first sub plot** tracks **training vs. validation loss** over 10 epochs. Training loss decreases slightly, while validation loss increases after a few epochs, indicating the possibility of **overfitting**—where the model fits well on the training data but struggles to generalise to unseen data. The **second sub plot** shows the **training vs. validation accuracy**, with both accuracies starting around **40%** and diverging as training progresses, with validation accuracy decreasing slightly over time. This divergence further supports the observation of overfitting.

In summary, while the model performs reasonably well on the validation set, the increasing validation loss and decreasing validation accuracy indicate that the model struggles with generalisation, and further tuning—such as adjusting **dropout rates or adding regularisation**—may be needed to improve performance and prevent overfitting.

Table 5: custom FFN Evaluation on validation set

	precision	recall	f1-score	support
accuracy			0.92	256234
Macro avg	0.92	0.92	0.92	256234
Weight avg	0.92	0.92	0.92	256234

Figure 5: The learning curve plot of Custom FFN on validation set

From **table 5**, the table summarises the performance metrics for the **custom FFN model** on the validation set, with **256,234 samples**. The model achieves an **accuracy of 92%**, indicating that it correctly classifies 92% of the validation samples. Both the **macro average** and **weighted average** for precision, recall, and F1-score are **0.92**, suggesting consistent performance across all classes with no significant imbalance issues. These metrics reflect that the model performs reliably and generalises well across the validation data.

Regarding the **figure 5**, the **first sub plot** illustrates the **training vs. validation loss** over 10 epochs. Both losses decrease sharply in the initial epochs, stabilising near **epoch 5** and remaining closely aligned throughout the training process, indicating that the model has converged without overfitting. The **second sub plot** tracks **training vs. validation accuracy** over the same period, showing rapid improvement in accuracy during the early epochs. Both training and validation accuracies align closely, maintaining a steady trajectory above **90%** by epoch 5. This consistent alignment between the curves indicates that the model generalises well to unseen data, avoiding both overfitting and underfitting.

Overall, the metrics and learning curves suggest that the custom FFN model is well-optimised, providing **high accuracy, precision, and recall** while maintaining stable performance throughout training and validation. This result confirms the robustness of the model architecture and its suitability for the given dataset.

FFN performance results

Baseline FFN performance results

Table 6A: baseline FFN Evaluation on test set 1

	precision	recall	f1-score	support
accuracy			0.86	50000
Macro avg	0.86	0.86	0.86	50000
Weight avg	0.86	0.86	0.86	50000

Table 6B: baseline FFN Evaluation on test set 2

	precision	recall	f1-score	support
accuracy			0.75	10000
Macro avg	0.78	0.75	0.75	10000
Weight avg	0.78	0.75	0.75	10000

The two tables above provide the **baseline FFN model's performance** on **Test Set 1** and **Test Set 2**, highlighting differences in metrics across the datasets.

In **Test Set 1**, the model achieves an **accuracy of 86%**, with both **macro and weighted averages** for precision, recall, and F1-score recorded at **0.86**. The consistency across these metrics suggests that the model performs well across all classes in this set, demonstrating a good balance between precision and recall for the **50,000 samples**.

However, the performance declines in **Test Set 2**, with the model reaching an **accuracy of 75%**. Both macro and weighted averages for precision, recall, and F1-score drop to **0.75**, indicating reduced effectiveness when evaluated on this set. With **10,000 samples**, the model might be encountering more challenging or less representative data, leading to this decrease in overall performance.

The comparison shows that while the baseline FFN performs well on the first test set, it struggles with the second set, possibly due to differences in the **distribution or complexity** of the data. This suggests that the model may require **additional tuning** or **regularisation** to improve its performance across varying datasets.

Final FFN performance results

Table 7A: Final FFN Evaluation on test set 1

	precision	recall	f1-score	support
accuracy			0.71	50000
Macro avg	0.75	0.71	0.71	50000
Weight avg	0.75	0.71	0.71	50000

Table 7B: Final FFN Evaluation on test set 2

	precision	recall	f1-score	support
accuracy			0.62	10000
Macro avg	0.67	0.62	0.61	10000
Weight avg	0.67	0.62	0.61	10000

The two tables above present the final FFN performance on Test Set 1 and Test Set 2, showing a notable difference in performance between the two datasets.

For Test Set 1, the model achieved an accuracy of 71%, with macro and weighted averages for precision, recall, and F1-score all at 0.75, 0.71, and 0.71. These results indicate that while the model maintains relatively balanced performance across classes, it still faced challenges in achieving higher predictive accuracy.

In Test Set 2, the performance dropped further, with an accuracy of 62%. The macro and weighted averages for precision, recall, and F1-score decreased to 0.67, 0.62, and 0.61,

respectively. This decline suggests that the model struggled more on this smaller dataset of 10,000 samples, potentially due to greater data variability or class imbalance, leading to reduced effectiveness.

To summarise, the comparison highlights the model's limitations in maintaining consistent performance across datasets. The drop in metrics between Test Set 1 and Test Set 2 suggests that further model tuning or additional regularisation could be vital to enhance generalisation and achieve better results across different data distributions.

Custom FFN performance results

Table 8A: Custom FFN Evaluation on test set 1

	precision	recall	f1-score	support
accuracy			0.86	50000
Macro avg	0.87	0.86	0.86	50000
Weight avg	0.87	0.86	0.86	50000

Table 8B: Custom FFN Evaluation on test set 2

	precision	recall	f1-score	support
accuracy			0.76	10000
Macro avg	0.78	0.76	0.76	10000
Weight avg	0.78	0.76	0.76	10000

The tables above show the **custom FFN model's performance** on **Test Set 1** and **Test Set 2**, highlighting its ability to generalise across different datasets.

In **Test Set 1**, the model achieves **86% accuracy**, with both **macro and weighted averages** for precision, recall, and F1-score at **0.87, 0.86, and 0.86**, respectively. This indicates that the model maintains strong, balanced performance across all classes for the **50,000 samples** in this dataset.

In **Test Set 2**, the model achieves an **accuracy of 76%**, with macro and weighted averages for precision, recall, and F1-score at **0.78, 0.76, and 0.76**. Although the performance decreases compared to Test Set 1, the drop is relatively modest, suggesting the model still generalises reasonably well to this smaller dataset of **10,000 samples**.

Overall, the custom FFN model demonstrates **strong performance and generalisation** across both datasets, with only a slight decline in metrics between Test Set 1 and Test Set 2.

This result indicates the **robustness of the custom architecture**, though further tuning could help improve performance on smaller or more complex datasets.

Evaluation comparison among three proposed FFN and CATBoost

The **custom FFN model** is likely to be the best classification architecture among the three FFN architectures as it consistently achieves high accuracy, precision, recall, and F1-score across both Test Sets 1 and 2. Furthermore, it demonstrates superior generalisation and robustness, with only a slight decline in performance on the smaller and potentially more challenging Test Set 2. These results suggest that the custom architecture and the adjustments applied (e.g., dropout, hyperparameters) were effective in improving the model's overall performance. The factors that affect generalisation ability would be thoroughly searched and identified in our advanced analysis. In comparison with the CATBoost **model**, **the custom FFN model** outperformed in computational feasibility for a big dataset, yet a bit fell short on generalising to the Test Set 2 (76%), while CATBoost slightly outperformed it by 1-2%.

Advanced analysis

Further investigation

In this section the performance of the FNN will be further investigated on the original ImageNet validation set and the ImageNetV2 validation set. The main focus will be to explore the hypothesis that model performance drops between Test Set 1 and Test Set 2 are not caused by overfitting but by complex images and noise in images in Test Set 2, consistent with Recht et al.'s (2019) analysis.

In order to do this the labels with the largest performance gap (in terms of either recall, precision or f1-score) between the validation sets were identified. A performance gap threshold of 0.6 in any of these metrics was set to identify and filter out the labels that caused the largest change in model performance between the validation sets. Note that a positive difference in corresponding performance metrics between the validation sets indicates that validation set 2 has a lower performance and then validation set 1 in this metric and vice versa.

The following figures shows the labels and corresponding difference in precision of the identified labels

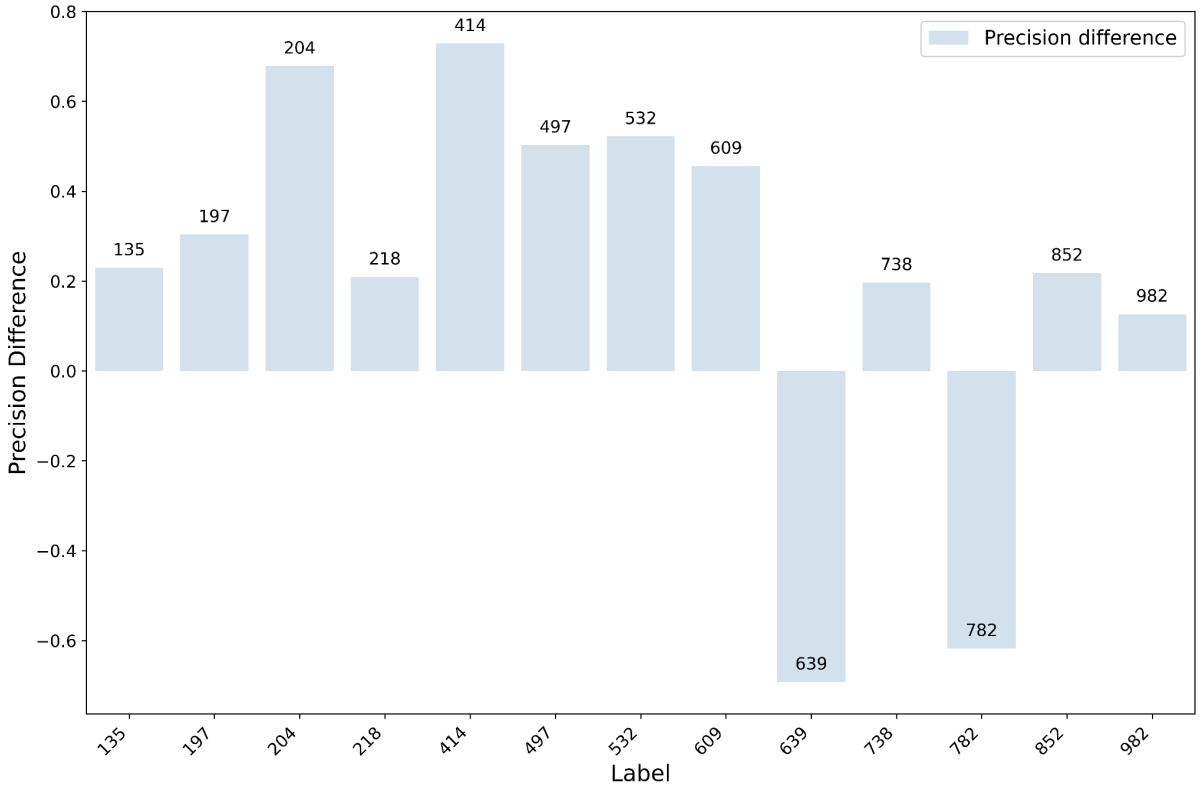


Figure 5: Difference in precision for labels that exceed performance threshold between the validation sets

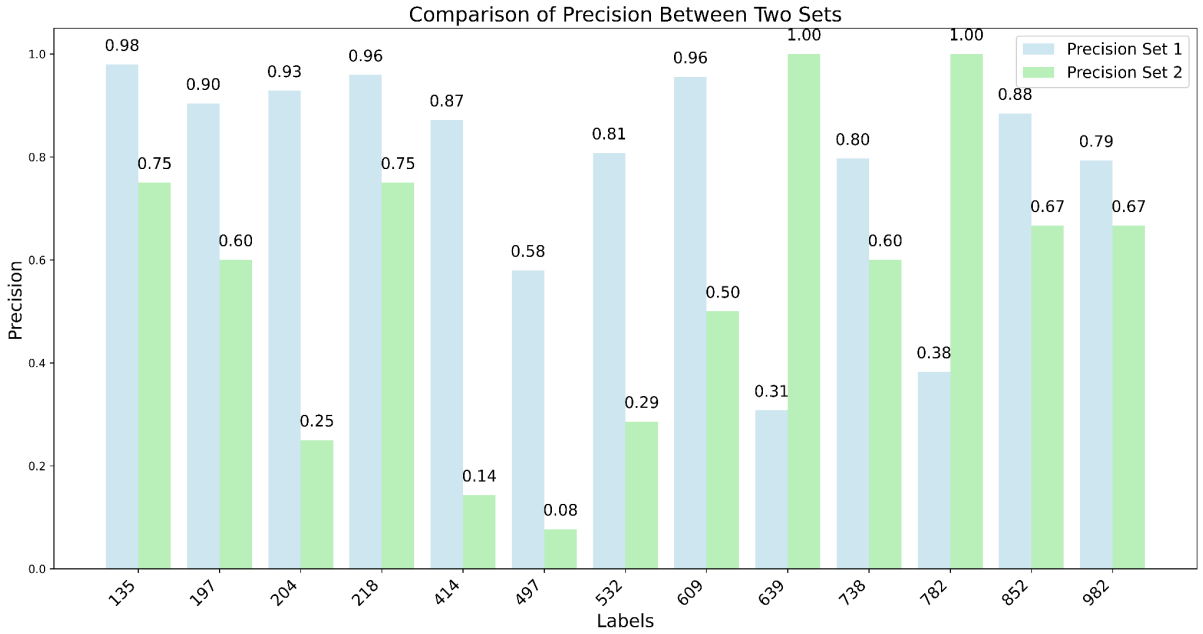


Figure 6: Comparison in precision for labels that exceed performance threshold between the validation sets

Furthermore, a comparison of the difference in recall is shown in the following figure

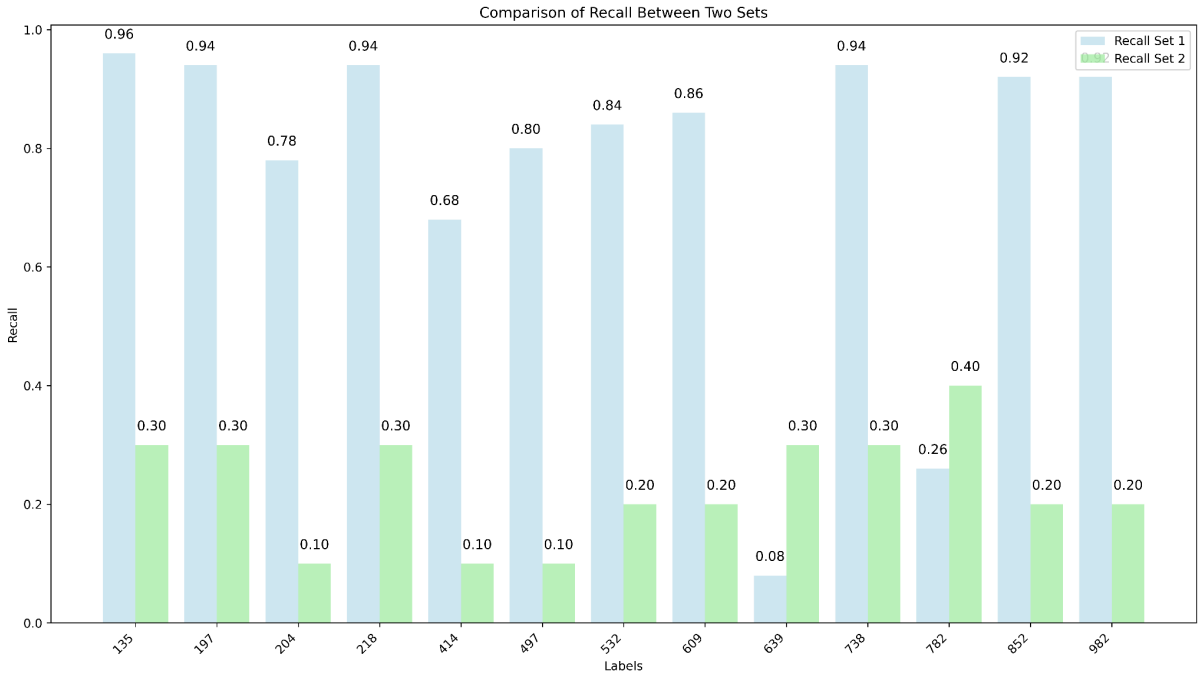


Figure 7: Comparison in recall for labels that exceed performance threshold between the validation sets

A comparison between the precision and recall metric is also shown in the plot below

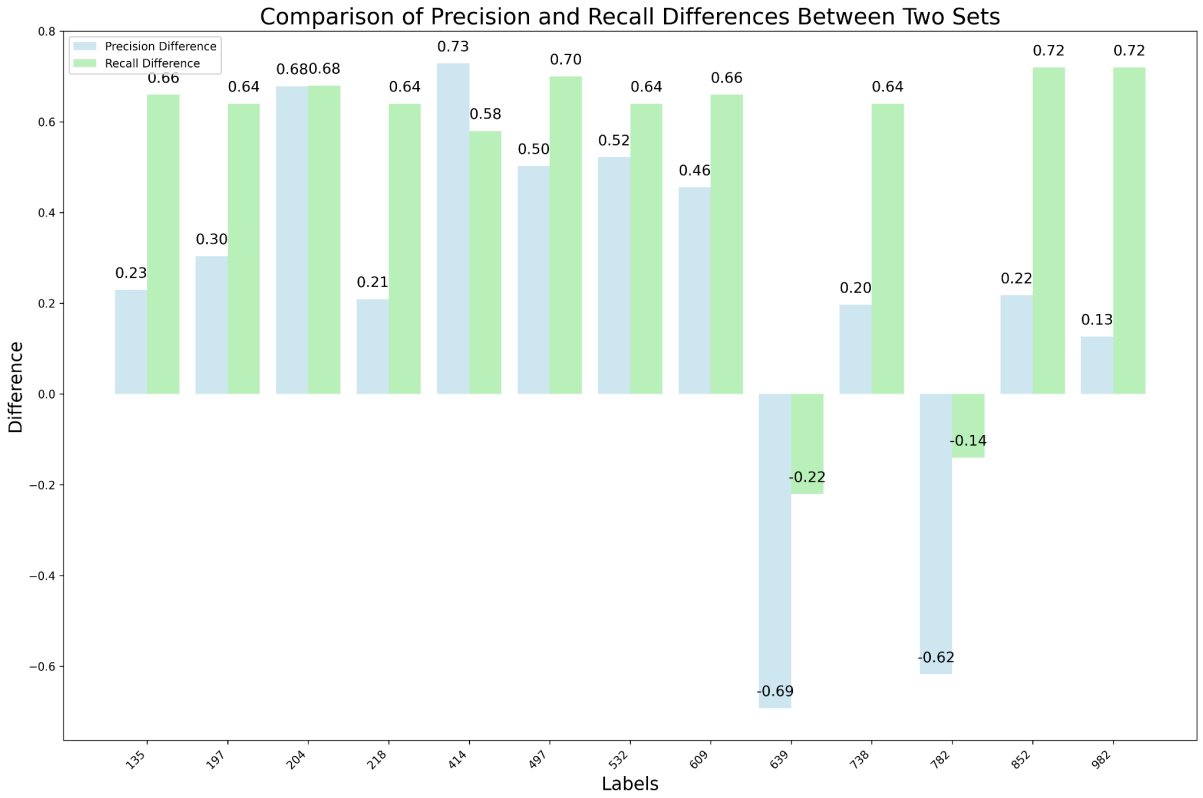


Figure 8: Comparison in recall and precision for labels that exceed performance threshold between the validation sets

Of particular interest is also that the recall difference stays consistently higher than the precision difference between the validation sets. This indicates that the model is more conservative in making a positive prediction, but in the cases where it does they are more likely to be correct.

For a more precise analysis, the 3 labels with the largest performance gap will be more thoroughly analysed. These labels are found to be “414, **backpacks**”, “497, **churches**” and “204, **Lhasas**” (It is worth noting that Lhasa refers to a specific breed of dog). To do this, specific images where the model outputs either a very low prediction probability for the correct class or a large prediction probability for an incorrect class are inspected.

for label 414, the following tables summarises this information for both validation set 1 and validation set 2:

	max_prob	prob_gap	pred_label	true label prob
4140	0.176808	-0.011953	762	2.842648e-02
4141	0.645309	-0.448368	652	9.426247e-04
4142	0.262794	-0.094246	879	5.301637e-03
4143	0.671885	-0.554686	608	1.325639e-05
4144	0.582791	-0.195435	636	3.873563e-01
4145	0.156942	-0.024047	772	1.328949e-01
4146	0.678139	-0.543231	414	6.781393e-01
4147	0.987587	-0.983243	874	1.510787e-10
4148	0.194243	-0.055783	785	5.365764e-07
4149	0.213940	-0.042869	68	7.546563e-02

	max_prob	prob_gap	pred_label	true label prob
20700	0.992063	-0.987002	414	9.920626e-01
20701	0.958624	-0.935340	636	9.597521e-03
20702	0.993427	-0.991652	414	9.934266e-01
20703	0.685749	-0.575021	414	6.857488e-01
20704	0.998733	-0.998118	414	9.987325e-01
20705	0.898590	-0.872324	414	8.985899e-01
20706	0.998511	-0.998189	414	9.985107e-01
20707	0.998873	-0.998483	414	9.988732e-01
20708	0.998556	-0.998216	414	9.985562e-01
20709	0.936524	-0.915277	636	3.364891e-04
20710	0.366316	-0.080271	414	3.663164e-01
20711	0.902352	-0.829108	414	9.023520e-01
20712	0.347084	-0.075727	879	4.079862e-02
20713	0.890360	-0.857177	899	6.325558e-12
20714	0.996444	-0.994625	414	9.964439e-01
20715	0.997261	-0.995735	414	9.972609e-01
20716	0.961570	-0.943383	804	3.152765e-11
20717	0.549653	-0.192510	748	4.306544e-02
20718	0.985085	-0.975607	414	9.850850e-01
20719	0.379592	-0.040548	414	3.795921e-01
20720	0.763862	-0.651372	414	7.638620e-01
20721	0.991060	-0.987554	414	9.910596e-01
20722	0.997356	-0.995231	414	9.973563e-01
20723	0.484077	-0.019967	373	4.641100e-01
20724	0.995874	-0.994862	414	9.958736e-01
20725	0.415971	-0.090073	608	4.787282e-03
20726	0.941490	-0.891335	414	9.414899e-01
20727	0.795909	-0.738301	414	7.959093e-01
20728	0.864959	-0.742195	414	8.649592e-01
20729	0.796714	-0.712685	414	7.967137e-01
20730	0.987580	-0.983856	414	9.875800e-01
20731	0.308970	-0.050085	414	3.089696e-01
20732	0.687100	-0.488262	748	1.988377e-01
20733	0.493172	-0.246434	414	4.931722e-01
20734	0.703183	-0.627365	636	3.653800e-02
20735	0.974507	-0.953559	414	9.745073e-01
20736	0.940438	-0.929765	414	9.404377e-01
20737	0.537018	-0.326116	414	5.370177e-01
20738	0.960999	-0.952915	414	9.609987e-01
20739	0.504583	-0.308149	658	2.425491e-03
20740	0.965773	-0.937078	414	9.657733e-01

20741	0.968619	-0.949313	748	1.930548e-02
20742	0.995982	-0.995002	414	9.959817e-01
20743	0.987457	-0.983477	414	9.874570e-01
20744	0.400847	-0.165470	655	1.746243e-01
20745	0.990050	-0.985782	414	9.900500e-01
20746	0.570214	-0.421202	838	8.525500e-03
20747	0.969826	-0.951250	414	9.698261e-01
20748	0.992830	-0.990826	797	2.003752e-03
20749	0.886053	-0.789792	523	6.764135e-10

Some images of particular interest are images 4147, 4148 and 4143. These images have the lowest true label probability, while also having a relatively large prediction probability for another label. The images are shown below:

Image 4147
misclassified as buss

Image 4148
misclassified as jeans

Image 4143
misclassified as seatbelt

The remaining misclassified images are:

Image 4140

Image 4141

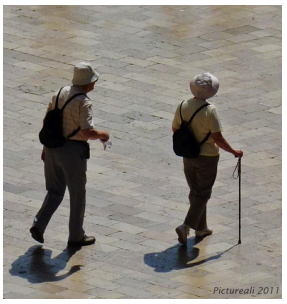


Image 4142

Image 4144

Image 4145

Image 4149

The only correctly identified image is 4146, which is shown below

Image 4146; correctly classified as backpack

For validation set 1, some interesting images that are highly misclassified are 20716 and 20713.

image 20716
misclassified as soap dispenser

Image 20713
misclassified as water jug

image 20706
correctly classified as backpack

Image 20702
correctly classified as backpack

for label 204, the following tables summarises this information for both validation set 1 and validation set 2:

	max_prob	prob_gap	pred_label	true label prob
2040	0.978597	-0.962374	155	0.016223
2041	0.428723	-0.320783	151	0.000023
2042	0.423893	-0.219636	153	0.084481
2043	0.335577	-0.118066	204	0.335577
2044	0.582340	-0.410022	152	0.020167
2045	0.361864	-0.121455	194	0.000736
2046	0.302214	-0.012160	968	0.001270
2047	0.742297	-0.511286	153	0.001011
2048	0.357712	-0.128530	154	0.002484
2049	0.902136	-0.848478	155	0.003906

	max_prob	prob_gap	pred_label	true label prob
10200	0.614609	-0.232506	155	0.382103
10201	0.172395	-0.006210	187	0.002863
10202	0.952934	-0.908018	155	0.044916
10203	0.981636	-0.972097	204	0.981636
10204	0.995069	-0.993849	204	0.995069
10205	0.996086	-0.994371	204	0.996086
10206	0.608280	-0.436739	153	0.049997
10207	0.343045	-0.092594	200	0.121350
10208	0.968180	-0.949298	204	0.968180
10209	0.964973	-0.946036	204	0.964973
10210	0.960042	-0.920875	204	0.960042
10211	0.998088	-0.996899	204	0.998088
10212	0.778365	-0.694525	204	0.778365
10213	0.988326	-0.982939	204	0.988326
10214	0.990484	-0.982619	204	0.990484
10215	0.991345	-0.983156	204	0.991345
10216	0.640931	-0.450141	204	0.640931
10217	0.982228	-0.965996	204	0.982228
10218	0.749020	-0.623455	204	0.749020
10219	0.852315	-0.708847	204	0.852315

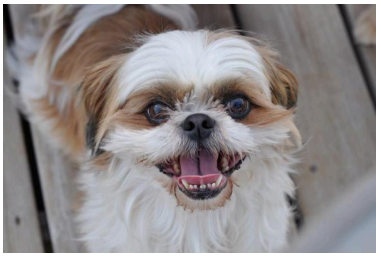
10220	0.925832	-0.868121	204	0.925832
10221	0.751801	-0.617074	204	0.751801
10222	0.997726	-0.996171	204	0.997726
10223	0.992723	-0.985689	204	0.992723
10224	0.919744	-0.872924	204	0.919744
10225	0.996703	-0.993739	204	0.996703
10226	0.580540	-0.320633	204	0.580540
10227	0.960912	-0.938477	204	0.960912
10228	0.930852	-0.863332	155	0.067520
10229	0.952883	-0.909090	204	0.952883
10230	0.991397	-0.985292	204	0.991397
10231	0.867675	-0.782154	204	0.867675
10232	0.793499	-0.668447	204	0.793499
10233	0.993708	-0.989007	204	0.993708
10234	0.951786	-0.922630	155	0.029155
10235	0.994113	-0.991044	204	0.994113
10236	0.858159	-0.769842	204	0.858159
10237	0.518824	-0.284593	153	0.234231
10238	0.925773	-0.880768	204	0.925773
10239	0.997681	-0.996726	204	0.997681
10240	0.779399	-0.578079	155	0.201320
10241	0.957657	-0.918417	204	0.957657
10242	0.525438	-0.269880	204	0.525438
10243	0.986721	-0.976067	204	0.986721
10244	0.966775	-0.936340	204	0.966775
10245	0.328586	-0.120772	266	0.005831
10246	0.878753	-0.803067	204	0.878753
10247	0.894784	-0.820039	204	0.894784
10248	0.502225	-0.190076	155	0.142603
10249	0.999132	-0.998599	204	0.999132

insert photo for validation set 1

Some interesting images to inspect manually are 2041, 2064, 2049.

2041: misclassified as a Shih-Tzu breed of dog

2064: misclassified as a cup



2049: misclassified as a Shih-Tzu breed of dog

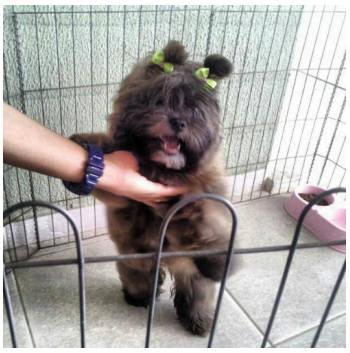


Image 4143; only image correctly classified as Lhases dog

Even to a human eye these images are easy to misclassify as a different dog breed, 2041 even seems to be incorrectly classified in the validation set itself as it does not resemble a Lhasas dog breed. These images continue to show a trend of images similar to, or containing different class imagery, being mistakenly classified by the model. In this case the misclassified examples are slightly different from those from label 414, as there is mainly imagery from one class in the image, however the images themselves do not strongly resemble the “Lhases” dog.

A selection of images from validation set 1 are shown below

Image 10203: correctly classified as Lhases

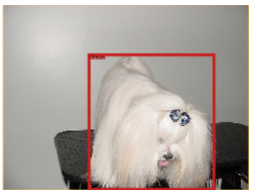
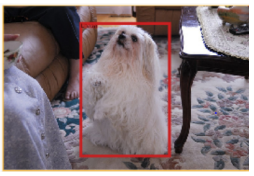


Image 10204: correctly classified as Lhases

Image 10248: incorrectly classified as Shih-Tzu



10201: Incorrectly classified as yorkshire terrier

For validation set 1 the model performs very well. In cases where it misclassifies it also tends to misclassify the image into one of the other dog breeds present in the dataset, with a particular tendency towards misclassifying as the "Shih-Tzu". It is somewhat surprising that the model manages to be as accurate as it is, given that some images are still somewhat hard to clearly classify as the "Lhases" dog. However, the vast majority of the images much more closely resemble a typical "Lhases" dog.

for label 497, the following tables summarises this information for both validation set 1 and validation set 2:

	max_prob	prob_gap	pred_label	true label prob
4970	0.295691	-0.079033	406	0.110001
4971	0.884443	-0.837992	857	0.000020
4972	0.726318	-0.623466	698	0.034243
4973	0.495977	-0.229650	442	0.010672
4974	0.612693	-0.267421	663	0.345272
4975	0.653521	-0.354831	884	0.000002
4976	0.966014	-0.937039	663	0.028975
4977	0.627194	-0.461938	442	0.042164
4978	0.617445	-0.395984	497	0.617445
4979	0.619493	-0.532895	492	0.000002

	max_prob	prob_gap	pred_label	true label prob
24850	0.551714	-0.419734	497	0.551714
24851	0.997497	-0.996777	497	0.997497
24852	0.980420	-0.966416	497	0.980420
24853	0.999508	-0.999412	497	0.999508
24854	0.986563	-0.982529	497	0.986563
24855	0.999700	-0.999626	497	0.999700
24856	0.997365	-0.996718	497	0.997365
24857	0.989700	-0.983419	406	0.006280
24858	0.797397	-0.741377	538	0.017512
24859	0.815675	-0.661009	497	0.815675
24860	0.986421	-0.974874	497	0.986421
24861	0.984854	-0.978868	497	0.984854
24862	0.992969	-0.991660	497	0.992969
24863	0.981886	-0.972836	497	0.981886
24864	0.983557	-0.979960	497	0.983557
24865	0.997996	-0.997441	497	0.997996
24866	0.994421	-0.989332	497	0.994421
24867	0.926066	-0.890138	497	0.926066
24868	0.843715	-0.719057	406	0.124657
24869	0.998431	-0.997563	497	0.998431
24870	0.885927	-0.804822	497	0.885927
24871	0.990751	-0.988414	497	0.990751
24872	0.972553	-0.947679	663	0.024875
24873	0.656647	-0.525731	497	0.656647
24874	0.903697	-0.814507	538	0.000001
24875	0.973604	-0.950190	497	0.973604
24876	0.959654	-0.934896	497	0.959654
24877	0.997311	-0.996535	497	0.997311
24878	0.991975	-0.989092	497	0.991975
24879	0.993979	-0.992211	497	0.993979
24880	0.646414	-0.537183	497	0.646414
24881	0.811025	-0.755475	497	0.811025
24882	0.999318	-0.999000	497	0.999318
24883	0.997326	-0.995629	497	0.997326
24884	0.938040	-0.918175	497	0.938040
24885	0.997248	-0.995252	497	0.997248
24886	0.987632	-0.983673	884	0.001596
24887	0.814883	-0.647812	406	0.167071
24888	0.992178	-0.988491	497	0.992178
24889	0.998251	-0.997781	497	0.998251
24890	0.993080	-0.989512	497	0.993080
24891	0.559034	-0.373330	497	0.559034
24892	0.999361	-0.999268	497	0.999361

24893	0.998262	-0.997635	497	0.998262
24894	0.909638	-0.826657	497	0.909638
24895	0.895559	-0.846258	884	0.000516
24896	0.978969	-0.974169	497	0.978969
24897	0.633057	-0.436029	663	0.011331
24898	0.528910	-0.395868	646	0.000063
24899	0.999906	-0.999858	497	0.999906

A subset of misclassified images, and the only correctly classified image, for validation set 2 are shown below

Image 4971: incorrectly classified as altar

Image 4979: incorrectly classified as chest

Image 4975: incorrectly classified as vault

Image 4972: incorrectly classified as palace

Image 4974: incorrectly classified as monastery

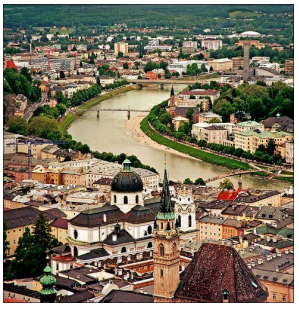


Image 4978: correctly classified as church

A subset of correct and incorrectly classified images for validation set 1 is shown below

Image 24852: correctly classified as church

Image 24850: correctly classified as church

Image 24874: misclassified as dome

Image 24895: misclassified as vault

Communicate Results

Visualisation evaluation

As seen from the manually inspected images above, a general trend for the model seems to be that images containing close to only imagery related to the correct class are classified correctly. While images that contain multiple elements related to different classes within the image, cause the model to misclassify more. Essentially, the presence of other classes within the image tends to overshadow the main class that is associated with the label. Based on visual inspection of the subset of classes above, this problem seems to be much more severe for validation set 2 then for validation set 1. For classes such as “backpack”, only three of the images in the validation set contain only a backpack in the image, the rest contain a series of different classes within every image. For some of the images, the main object also seems to be that of a different class indicating that the image could potentially be mislabeled. If this problem is to be present across a multitude of classes in validation set 2, then it would naturally be a large cause of the resulting performance drop of the model. It is worth noting that in cases where validation set 1 contains images with elements related to different classes, it also experiences similar misclassifications. However, validation set 1 seems to contain much more of these images. Based on this, the performance gap is very sensible.
Another trend is that certain image classes are very closely related to each other, such as the Shih-Tzu dog and the Lhases dog which can be hard to distinguish even for a human. This seems to cause misclassification in both validation set 1 and validation set 2. However, validation set 1 seems to contain images that more closely resemble a typical “Lhases” dog.

In short, a visual inspection of a subset of images of classes with significant performance gaps between the validation sets provides evidence towards our initial hypothesis. The performance gap appears to be driven by the presence of complex and noisy images, which appear to be much more prevalent in validation set 2, aligning with the analysis by Recht et

al. (2019). This is especially shown by the presence of multiple classes within the image overshadowing the correct label of the class, a problem which seems much more prevalent for validation set 2.

Suggestions

Our results support our original hypothesis: classifiers trained exclusively on the ImageNet dataset tend to become **biassed towards the specific characteristics of that dataset**, leading to reduced performance when evaluated on different validation sets. This overfitting occurs because the model learns patterns unique to the ImageNet images, which may not generalise well to other datasets with different distributions.

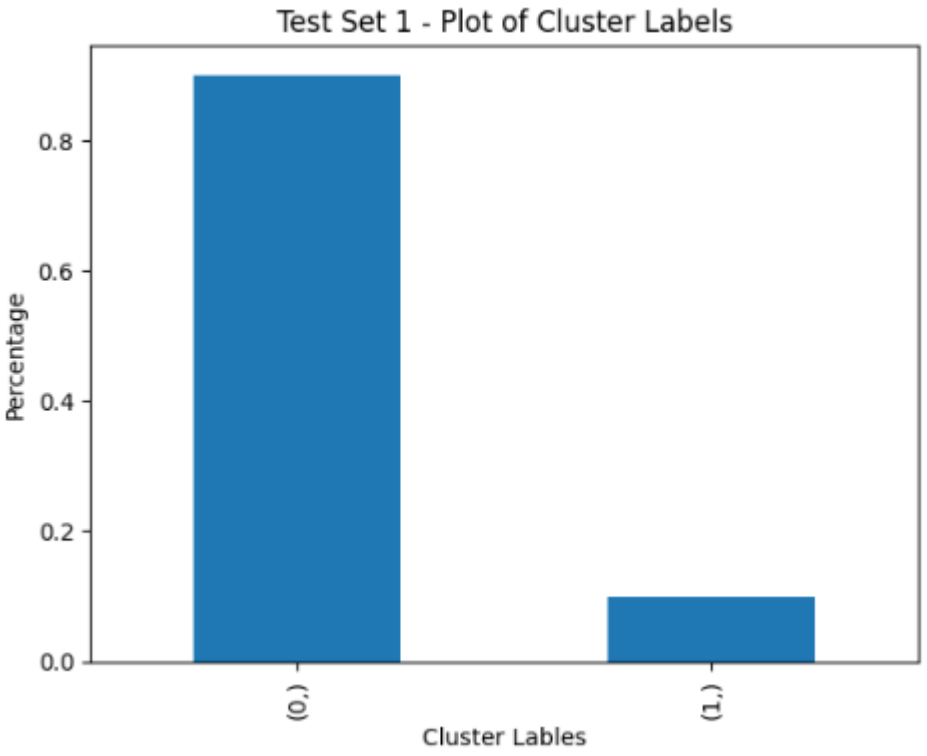
Previous analyses highlighted a significant difference between test set 1 and test set 2. **Test set 2 contains more challenging and complex images**—often featuring multiple objects within a single image—whereas test set 1 typically includes images focusing on a single object. These variations in image backgrounds, themes, and complexity likely contribute to the observed performance disparity between the two test sets.

To investigate the underlying reasons for this performance gap, we conducted a clustering analysis on both test sets. We hypothesised that if test set 1 consists of simpler images, clustering would reveal a dominant cluster representing this homogeneity. Conversely, test set 2, with its increased complexity, would exhibit a more even cluster distribution.

We utilised **MiniBatchKMeans** for clustering due to its computational efficiency over traditional KMeans. By processing small, random batches of data, MiniBatchKMeans significantly reduces computation time while maintaining clustering effectiveness—a crucial consideration given our large datasets and the need for rapid iteration.

Clustering Results and Implications

For **test set 1**, the clustering label distribution revealed that **cluster label 0 accounts for over 85%** of all records, while **cluster label 1 comprises approximately 15%**. This skewed distribution indicates that the majority of images in test set 1 fall into a single cluster, suggesting that the dataset contains images that are relatively homogeneous in theme and complexity. This can be seen in the plot below.

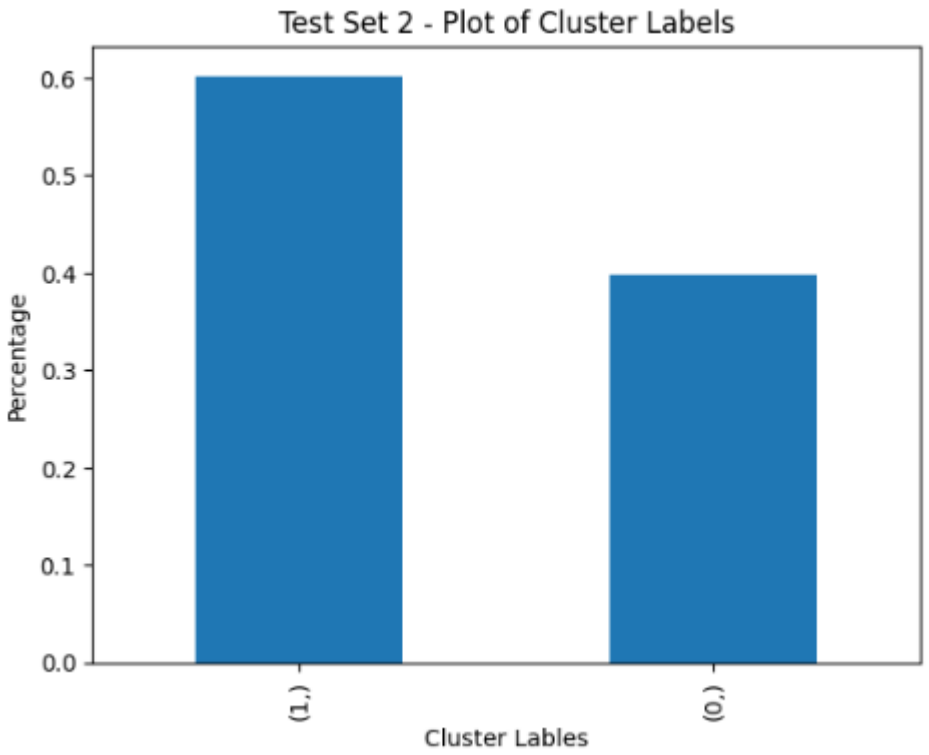


The dominance of one cluster supports our initial assumption that test set 1 consists predominantly of simpler images with less variation. This homogeneity likely contributes to the classifier's higher performance on this set due to several factors:

- **Consistent Feature Representation**: With most images sharing similar characteristics, the features extracted by the model are more uniform, making it easier for the classifier to learn and recognize patterns.
- **Reduced Complexity**: Simpler images with fewer objects and less intricate backgrounds reduce the cognitive load on the model, allowing for more accurate predictions.
- **Lower Intra-Class Variability**: The minimal variation within classes means that the model doesn't need to account for a wide range of variations, enhancing its predictive capabilities.

This clustering result implies that the **training data may have been more representative of the types of images found in test set 1**, further explaining the higher accuracy observed.

In contrast, the clustering results for **test set 2** showed a different pattern. Approximately **60% of records belong to cluster label 0**, while the remaining **40% are assigned to cluster label 1**. This more balanced cluster distribution implies that test set 2 contains images that are more diverse and complex, encompassing multiple themes and objects. This can be seen in the plot below.



The near-even split between the two clusters in test set 2 suggests several important implications:

- **Increased Diversity**: The images in test set 2 likely cover a broader range of subjects, styles, and complexities, including varying lighting conditions, angles, and backgrounds.
- **Higher Complexity**: Images containing multiple objects or intricate details introduce additional features that the classifier must process, increasing the difficulty of accurate classification.
- **Domain Shift**: The differences between the images in test set 2 and those the model was trained on may represent a **domain shift**, where the statistical properties of the input data change between the training and test phases.

These factors contribute to the classifier's lower accuracy on test set 2. The model may struggle to generalise from the training data if it hasn't encountered similar levels of complexity and diversity during training.

Conclusion and Recommendations

The clustering analysis reveals a fundamental issue: our classifier excels on data that is similar to what it was trained on (test set 1) but struggles with data that deviates in complexity and diversity (test set 2). This disparity suggests that the model lacks robustness and generalisation capabilities necessary for handling more complex real-world data. The observed **domain shift** between the training data and test set 2 likely contributes to the lower accuracy, as the statistical differences challenge the model's ability to generalise.

To reduce the performance gap and address the domain shift, we recommend several strategies:

1. **Enhance Training Data Diversity:** Incorporate complex images into the training dataset to better represent the diversity found in test set 2. Balancing the training data with a mix of simple and complex images can prevent the model from overfitting to simpler patterns. This can be achieved by sourcing additional datasets that include varied scenes, objects, and contexts. Including images with different lighting conditions, backgrounds, and occlusions will expose the model to a wider range of scenarios. By enriching the training data, the model learns to generalise better and becomes more robust when encountering unfamiliar or intricate images during testing.
2. **Data Augmentation:** Use data augmentation techniques such as rotation, scaling, translation, and adding noise to create variations of existing images when training the computer vision model. This can simulate complexity, increase robustness, and help the model generalise to unseen data. Additionally, applying random flips, brightness adjustments, contrast changes, and colour jittering can further enhance the diversity of the training set. Implementing augmentation not only increases the amount of training data but also helps the model become invariant to common image transformations, which is especially beneficial for complex images in test set 2.
3. **Adjust Model Complexity:** Adjusting the layer depth and hyperparameters of the MLP can allow the model to learn higher-level abstractions. We suggest making our MLP deeper by adding additional hidden layers. Moreover, experimenting with different numbers of neurons per layer and using advanced activation functions like ReLU, Leaky ReLU, or ELU can improve the network's ability to capture non-linear patterns. Incorporating regularisation techniques such as dropout and batch normalisation can help prevent overfitting as the model's complexity increases. Fine-tuning the learning rate and trying different optimizers like Adam or RMSprop may also enhance training efficiency and convergence.
4. **Feature Engineering:** Explore additional feature extraction methods that can capture the nuances of complex images. This could be in the form of metadata of the images, such as the camera model, exposure settings, or geolocation data. Incorporating texture descriptors, edge detection features, or colour histograms can provide more detailed information for the model to utilise. Utilising techniques like feature selection or dimensionality reduction (e.g., t-SNE, UMAP) can help in identifying the most informative features, thereby improving model performance on complex images.

5. **Domain Adaptation Techniques:** Employ transfer learning by fine-tuning the model on a subset of data similar to test set 2 to help it adapt to new patterns. Use domain-adversarial neural networks to reduce the discrepancy between training and test data distributions. Additionally, applying methods like Maximum Mean Discrepancy (MMD) or Correlation Alignment (CORAL) can align the feature representations between the source and target domains. Implementing unsupervised domain adaptation techniques when labelled data is scarce can also be beneficial. These approaches help the model become more resilient to domain shifts, improving its ability to perform well on diverse and complex datasets like test set 2.

We believe that these recommended approaches will help reduce the performance gap between the two test sets and enhance the classifier's generalisation capabilities. Our main finding from the cluster analysis was that a domain shift between the two sets is the leading cause behind the performance gap, as test set 2 contains more complex and real-world images. Addressing this issue by altering the dataset is challenging since it is beyond our control for this assignment.

Furthermore, we believe that making our MLP deeper by adding more layers—and importantly, incorporating additional dropout layers to reduce overfitting—will result in a more robust and reliable classifier that performs consistently across both datasets. However, the lack of funds to purchase computational resources like GPUs on Google Colab has prevented us from retraining a deeper version of our MLP on the entire dataset. All attempts to train this on our local computers with CPUs have resulted in system crashes due to resource limitations. These constraints may have limited the model's capacity to learn complex patterns necessary for improved generalisation.

In future work, we plan to seek access to enhanced computational resources, such as GPUs or cloud-based computing platforms, to implement deeper neural networks. Additionally, we will explore unsupervised domain adaptation techniques. These steps aim to create a more robust and reliable classifier that performs consistently across different datasets, ultimately enhancing its applicability to real-world scenarios.

Lessons Learned and Takeaways

Throughout this project, we have gained several important insights that will inform our future work in machine learning and big data analytics.

The Role of Baseline Datasets

We recognize the crucial role that baseline datasets play in building and evaluating models in the machine learning space. They provide a solid reference point, allowing us to compare our model's performance against established benchmarks and plan our next steps accordingly. However, this task has also highlighted the risks of overreliance on baseline datasets. Excessive dependence can lead to biassed results, where the model becomes overfitted to the baseline data and performs poorly when deployed in real-world scenarios, which are often more diverse, complex, and dynamic. This underscores the importance of ensuring that our models are trained and validated on data that accurately reflects the environments in which they will be used.

Importance of Algorithm Selection in Big Data Experiments

Another key lesson is the significance of algorithm choice when conducting big data experiments. Running models on large datasets can result in significant training times, which may hinder rapid development and iteration. We learned the value of starting with a subset of the data to perform quick experiments. By initially testing our models on smaller datasets, we can identify promising approaches and refine our models before scaling up to train on the entire dataset. This aligns with the philosophy of "fail fast and fail early," which is critical for efficient and effective machine learning model development.

Consideration of Computing Resources

We also learned about the critical role of computing power in big data tasks. The choice between using CPUs and GPUs can significantly impact the feasibility and performance of different machine learning models. This experience has taught us to carefully consider the available hardware when planning and conducting big data projects. As students, it's easy to become overly focused on the software aspects, such as selecting and tuning models, while overlooking the hardware constraints that can affect our ability to execute these models efficiently. In future projects, we will ensure that hardware considerations are integrated into our planning process.

Understanding Domain Shift and Model Generalization

Our project highlighted the challenges posed by domain shift—the differences between training data and real-world data. We learned that models trained on datasets that do not fully represent the diversity and complexity of real-world scenarios may perform poorly when deployed. This emphasises the importance of ensuring that training data is comprehensive and that models are robust enough to generalise beyond the training set. Incorporating diverse and

complex data into training can improve model adaptability and performance in varied environments.

The Value of Clustering and Exploratory Data Analysis

By employing clustering techniques, we gained deeper insights into the underlying structure of our datasets. This exploratory data analysis allowed us to identify significant differences between test sets, informing our understanding of why the model performed differently across them. We learned that such techniques are invaluable for diagnosing issues and guiding model improvement strategies. Effective data exploration can uncover patterns and anomalies that might otherwise remain hidden.

Bibliography

Recht, B., Roelofs, R., Schmidt, L., & Shankar, V. (2019). Do ImageNet classifiers generalise to ImageNet? *Proceedings of the 36th International Conference on Machine Learning (ICML)*. https://arxiv.org/abs/1902.10811

Analytics Vidhya. (2024). What is XGBoost algorithm and how does it work? Retrieved from Analytics Vidhya

IBM. (n.d.). What is XGBoost? Retrieved from IBM

Machine Learning Models. (2024). A powerful ML model for classification and regression. Retrieved from Machine Learning Models

CatBoost. (n.d.). Open-Source Gradient Boosting Library. Retrieved from CatBoost

Built In. (2024). What Is CatBoost?. Retrieved from Built In

Analytics Vidhya. (2024). An Introduction to Logistic Regression. Retrieved from Analytics Vidhya

IBM. (n.d.). What Is Logistic Regression?. Retrieved from IBM

IBM. (n.d.). What Is Random Forest?. Retrieved from IBM

Analytics Vidhya. (2024). Understanding Random Forest Algorithm. Retrieved from Analytics Vidhya

Friedman, J.H., 2001. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5), pp.1189-1232.

Goodfellow, I., Bengio, Y. & Courville, A., 2016. *Deep Learning*. MIT Press.

LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P., 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), pp.2278-2324.

LeCun, Y., Bengio, Y. & Hinton, G., 2015. Deep learning. *Nature*, 521(7553), pp.436-444.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A.V. & Gulin, A., 2018. CatBoost: unbiased boosting with categorical features. *Advances in Neural Information Processing Systems*, 31.

MyScale. (2024, April 24). *Maximizing training efficiency with TensorFlow GPU: A step-by-step guide*. MyScale. https://myscale.com/blog/maximize-training-efficiency-tensorflow-gpu-step-by-step-guide/

Neptune.ai. (2024). *When to choose CatBoost over XGBoost or LightGBM*. Neptune.ai. https://neptune.ai/blog/catboost-vs-xgboost-vs-lightgbm

Scaler. (2024). *Accelerating deep learning with TensorFlow GPU*. Scaler. https://www.scaler.com/topics/accelerating-deep-learning-with-tensorflow-gpu/

DataCamp. (2024). *Google Colab tutorial for data scientists*. Retrieved from https://www.datacamp.com/tutorial/google-colab

McCormick, C. (2024). *Colab GPUs features & pricing*. Retrieved from https://mccormickml.com

Kaggle. (2024). A comparison of Optuna and RandomizedSearchCV. https://www.kaggle.com/code/unworried1686/a-comparison-of-optuna-and-randomizedsearchcv

Keylabs. (2024). Grid search, random search, and Bayesian optimization. https://keylabs.ai

Hashnode. (2024). *Optuna vs Hyperopt: Which hyperparameter optimization library should you choose?* https://hashnode.com