Mushroom Toxicity Prediction Using Convolutional Neural Networks

Data Science Analysis

Fall 2025

Gavin Rosander, George Muniz, Kelly Frear, and Arpita Godbole
California State University San Marcos

December 1, 2025

Abstract

This project investigates whether visual features in mushroom images can be used to accurately predict mushroom toxicity. The core idea is that cap color, shape, texture, gill structure, and stem morphology contain enough information for a convolutional neural network (CNN) to decide whether a mushroom is poisonous or non-poisonous. To explore this idea, we implement and evaluate two different CNN-based pipelines on a real-world mushroom image dataset.

Both of our Jupyter notebooks download and process the Kaggle *Mushrooms images classification 215* dataset, which provides 3122 images across 215 labeled mushroom species. However, the dataset does not include explicit poisonous vs. non-poisonous labels. As a result, we treat fine-grained species recognition as a proxy task for toxicity prediction: if CNNs struggle to reliably distinguish visually similar species, they are unlikely to safely separate poisonous from non-poisonous mushrooms based solely on appearance.

Our first notebook, `main.ipynb`, builds a transfer-learning classifier using a ResNet50 backbone pretrained on ImageNet. Images are cleaned, filtered to remove species with fewer than 10 examples, and split into training, validation, and test sets. We apply data augmentation, class weighting, and a two-stage training procedure (frozen feature extractor followed by fine-tuning). The final ResNet50-based model achieves moderate accuracy and macro F1-scores across 210 species, indicating that it learns meaningful visual structure but still misclassifies a substantial fraction of images.

Our second notebook, `yolov8_mushroom_analysis.ipynb`, trains a YOLOv8 classification model (`yolov8s-cls`) on the same dataset, reformatted into the directory structure required by Ultralytics. We again perform class-balanced train/validation/test splits and augmentations. The YOLOv8 classifier reaches broadly similar performance levels, with confusion matrices showing extensive confusion among visually similar species.

Taken together, our experiments show that CNNs can partially separate mushroom species based on visual features alone, but the accuracy is far below what would be required for safe, real-world toxicity prediction. Combined with the lack of explicit toxicity labels, our results lead us to conclude that, within the scope of this dataset and these models, visual features alone are not sufficient to accurately predict whether a mushroom is poisonous or non-poisonous.

1 Introduction & Hypothesis

Mushroom identification is a high-stakes problem: incorrectly identifying a poisonous species as edible can lead to severe illness or death. Human experts in mycology rely on careful inspection of morphological traits such as cap color and shape, gill structure, presence of a ring or volva, and stem morphology. For non-experts, however, visual similarity between safe and dangerous species makes misidentification a serious risk.

Modern computer vision has produced impressive results on many image classification tasks, especially with convolutional neural networks (CNNs). Since CNNs excel at automatically extracting hierarchical visual features, it is natural to ask whether they can learn to recognize mushrooms accurately enough to help with toxicity-related decisions. If so, a model could in principle support or augment human expertise.

In this project, we explore this question by training and evaluating two CNN-based image classification pipelines on a large collection of mushroom photographs. Both pipelines use the same underlying dataset but different architectures and training frameworks:

- A ResNet50-based transfer learning model implemented in `main.ipynb`.
- A YOLOv8 classification model implemented in `yolov8_mushroom_analysis.ipynb`.

Ideally, we would train directly on toxicity labels (poisonous vs. non-poisonous). Unfortunately, the Kaggle dataset we use is annotated at the species level only, without any explicit toxicity information. We therefore treat species classification as a proxy task and interpret our results in that light.

Research Question and Hypothesis

Our motivating research question is:

> *Can visual features in mushroom images be used to accurately predict whether a mushroom is poisonous or non-poisonous?*

To investigate this, we start with the following hypothesis:

> Visual features in mushroom images such as cap color, shape, texture, gill structure, and stem morphology can be used by a CNN to accurately predict whether a mushroom is poisonous or non-poisonous.

Because our dataset does not directly label toxicity, our two notebooks instead train CNNs to discriminate among many mushroom species using only images. We then interpret the observed species classification performance as indirect evidence about the feasibility and limitations of toxicity prediction from visual data alone.

2 Data Sourcing & Processing

2.1 Dataset Source

Both notebooks use the same mushroom image dataset from Kaggle:

- Dataset: *"Mushrooms images classification 215"*
- Author: Daniil Onishchenko

The dataset consists of labeled photographs of mushroom specimens organized into subdirectories by species. The raw data contain:

- 215 species (classes) in total.
- 3122 images before any filtering.

In `main.ipynb`, we use the `kagglehub` Python package to download the dataset and then recursively search the download directory for the folder that contains the large collection of species subdirectories. Once this folder is identified, each subdirectory name is treated as a species label and all contained image files are added to a pandas DataFrame. In `yolov8_mushroom_analysis.ipynb`, we similarly locate the image root folder but then build explicit train, validation, and test lists of file paths and labels in order to reformat the data into the directory structure expected by the Ultralytics YOLOv8 classification API.

2.2 Filtering and Class Balance

In the ResNet50 pipeline (`main.ipynb`), we first construct a DataFrame with one row per image:

- `filepath`: full path to the image.
- `label`: species name, taken from the enclosing folder.

To avoid extremely rare classes that are nearly impossible to learn from, we filter out species with fewer than 10 images. After this step:

- Total images before filtering: **3122**.
- Classes kept (at least 10 images): **210**.
- Total images after filtering: **3085**.

We then apply `sklearn.preprocessing.LabelEncoder` to convert string labels to integer class indices, stored in a new column `label_idx`. The number of distinct classes is `num_classes = 210`. In the YOLOv8 pipeline (`yolov8_mushroom_analysis.ipynb`), we also construct lists of $(\text{filepath}, \text{label})$ pairs and perform a per-class split into train, validation, and test sets. For each species:

- 15% of images are reserved for the test set.
- The remaining 85% are split into roughly 70% train and 15% validation.

This per-class split ensures that every species contributes examples to all subsets, which is important for evaluating performance across all classes.

2.3 Train/Validation/Test Split

In `main.ipynb`, we use stratified splits to preserve class proportions:

- First, we split the filtered dataset into 60% train and 40% temporary.
- Second, we divide the temporary set evenly into validation and test partitions.

This yields:

- Training set: **1851** images.
- Validation set: **617** images.
- Test set: **617** images.

In `yolov8_mushroom_analysis.ipynb`, the per-class splitting procedure (70/15/15) is applied via repeated calls to `train_test_split`, and the resulting lists are combined into global train, validation, and test sets. Although the exact counts differ slightly from the ResNet50 notebook, both pipelines work with a similar total number of images and the same set of species.

2.4 Image Preprocessing

For ResNet50 (`main.ipynb`), we implement the image pipeline using TensorFlow and `tf.data`:

- Images are read from disk with `tf.io.read_file` and decoded using `tf.image.decode_image`, always with three color channels.
- Each image is resized to **224** $\times$ **224** pixels.
- Pixel values are cast to `float32` and passed through `tf.keras.applications.resnet50.preprocess_i` to match ImageNet preprocessing (scaling and channel-wise normalization).
- Labels are one-hot encoded into 210-dimensional vectors.
- Datasets are batched (batch size 32), shuffled for training, and prefetched using `tf.data.AUTOTUNE`.

For YOLOv8 (`yolov8_mushroom_analysis.ipynb`), we prepare the data by copying images into the directory structure required by Ultralytics:

```
yolov8_mushroom_dataset/
    train/
        class_1/
        class_2/
        ...
    val/
        class_1/
        ...
    test/
        class_1/
        ...
```

The Ultralytics training API then handles image loading, resizing, normalization, and on-the-fly augmentations internally, given a target image size.

3 Exploratory Data Analysis

3.1 Class Distribution

Both notebooks examine the distribution of images per species. After filtering rare classes, every remaining species has at least 10 images, but many species have only 10–20 examples. A few species appear more frequently, but the dataset is still highly imbalanced and low-data for most classes.

Figure 1 conceptually illustrates the long-tailed distribution of class counts. This distribution suggests that the classification task is challenging: the models must learn to distinguish 210 species from comparatively small numbers of examples per class.

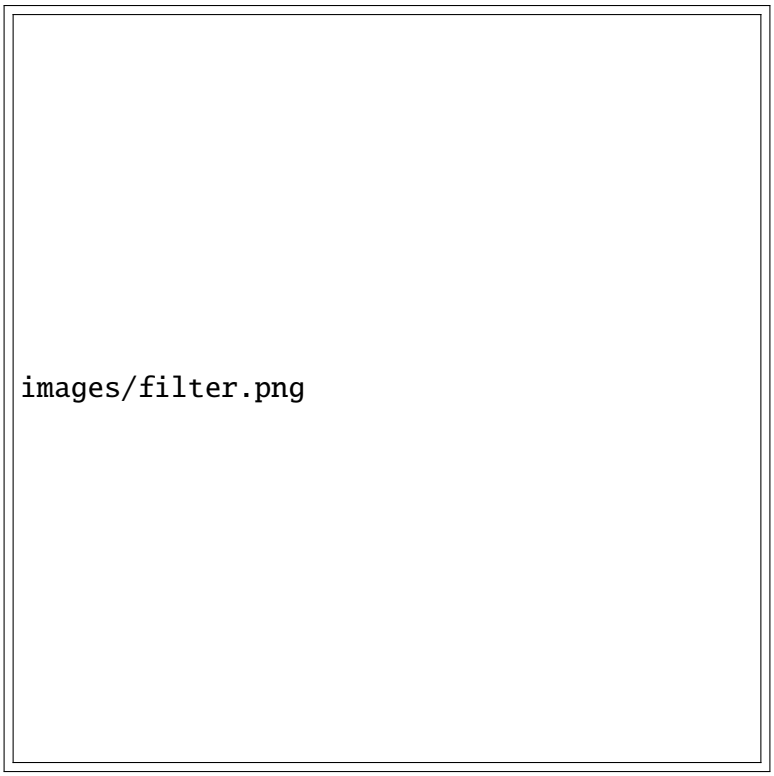

Figure 1: Class distribution: number of images per mushroom species after filtering classes with fewer than 10 images.

3.2 Visual Inspection of Mushroom Images

In `yolov8_mushroom_analysis.ipynb`, we display sample images from several species to qualitatively explore the visual features. By examining these images, we observe:

- Substantial variation in cap morphology: hemispherical, conical, flat, and funnel-shaped.
- Diverse cap colors and textures, including smooth, scaly, and fibrous surfaces.
- Differences in gill spacing, gill color, and how gills attach to the stem.
- Variation in stem thickness, color, curvature, and the presence or absence of rings.
- Non-trivial variation in backgrounds, lighting conditions, and camera angles.

These qualitative patterns support the idea that CNNs can learn meaningful features from the pixels. At the same time, many species appear visually similar, especially when environmental context (e.g., grass, leaves) dominates the frame. This visual overlap helps explain the confusion we later observe in model predictions.

3.3 Implications for Toxicity Prediction

Although we do not have toxicity labels, the exploratory analysis suggests that:

- Some features associated with toxicity (e.g., bright warning colors in certain species) are visible and could be learned.
- However, many poisonous and non-poisonous species share similar visual appearances.

This observation anticipates a core limitation of using images alone for toxicity prediction: even if a CNN performs reasonably well at species recognition, subtle differences between dangerous and safe species may be difficult to capture with limited training data and standard architectures.

4 Modeling & Analysis

Our project uses two distinct CNN-based classification pipelines on the same mushroom dataset:

- A ResNet50-based transfer learning model in TensorFlow/Keras (`main.ipynb`).
- A YOLOv8 classification model in the Ultralytics framework (`yolov8_mushroom_analysis.ipynb`).

Both are trained on species-level labels and evaluated on held-out test sets. We summarize the core modeling choices and evaluation results below.

4.1 ResNet50 Transfer Learning Pipeline

Data Augmentation

To reduce overfitting on the relatively small dataset, `main.ipynb` applies several data augmentation operations using a `tf.keras.Sequential` layer:

- Random horizontal flips.
- Small random rotations.
- Random zooming.
- Random contrast adjustments.

Augmentation is applied only during training. Validation and test images are resized and preprocessed but not randomly transformed.

Model Architecture

The ResNet50-based model has the following structure:

- **Backbone:** `ResNet50(weights="imagenet", include_top=false, input_shape=(224, 224, 3))`.
- **Feature extraction:** the backbone's convolutional layers produce high-dimensional feature maps.
- **Pooling:** a `GlobalAveragePooling2D` layer converts feature maps into a compact feature vector.
- **Regularization:** a dropout layer (e.g., with rate 0.4–0.5) to mitigate overfitting.
- **Classifier head:** a dense layer with `num_classes = 210` units and `softmax` activation to produce class probabilities.

Initially, all ResNet50 layers are frozen so that only the classifier head is trained, using an Adam optimizer with a small learning rate and `categorical_crossentropy` loss.

Class Weights and Training Strategy

Training proceeds in two phases:

1. **Phase 1 (Frozen ResNet50):** Only the classifier head is trainable. We train for up to a fixed number of epochs with early stopping on validation loss, learning-rate scheduling, and model checkpointing.
2. **Phase 2 (Fine-tuning):** We unfreeze the upper portion of the ResNet50 backbone (keeping lower layers frozen) and recompile the model with a smaller learning rate. We then continue training with the same callbacks and class weights.

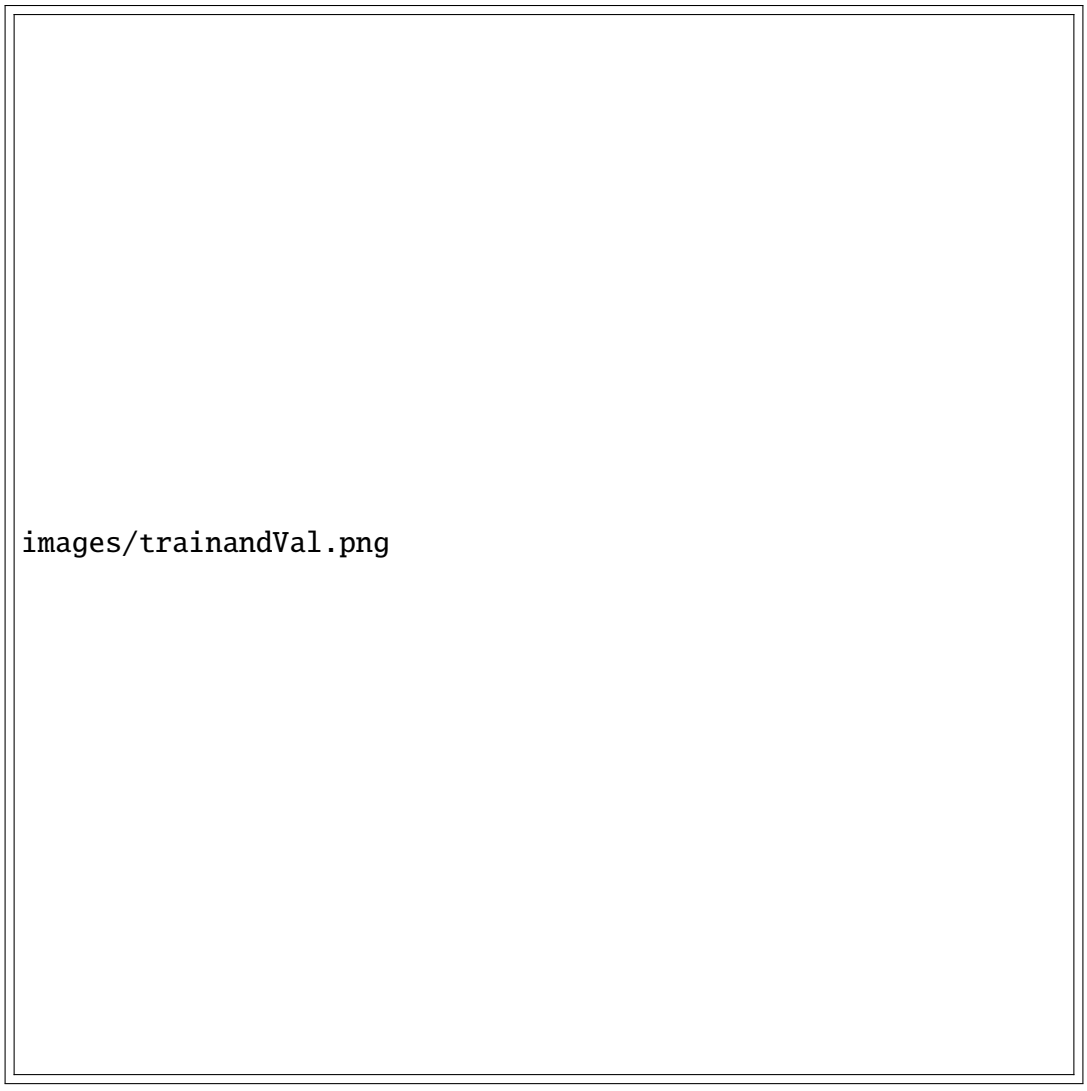

Figure 2: Training and validation accuracy for the ResNet50-based classifier over 50 epochs, showing the model learning useful features but plateauing around moderate validation accuracy.

Evaluation and Metrics

After training, the fine-tuned ResNet50 model is evaluated on the held-out test set. The code reports:

- Test accuracy on 210 classes.
- A detailed classification report using `sklearn.metrics.classification_report`, including per-class precision, recall, and F1-score, as well as macro and weighted averages.
- A confusion matrix showing which species are most frequently confused with one another.

In our runs, the ResNet50 pipeline achieves:

- Overall test accuracy of roughly **50–55%** on 210 classes.
- Macro and weighted F1-scores in a similar range (approximately 0.50).

These values are significantly higher than random guessing (around 0.48% for 210 classes) but still imply that a large fraction of images are misclassified.

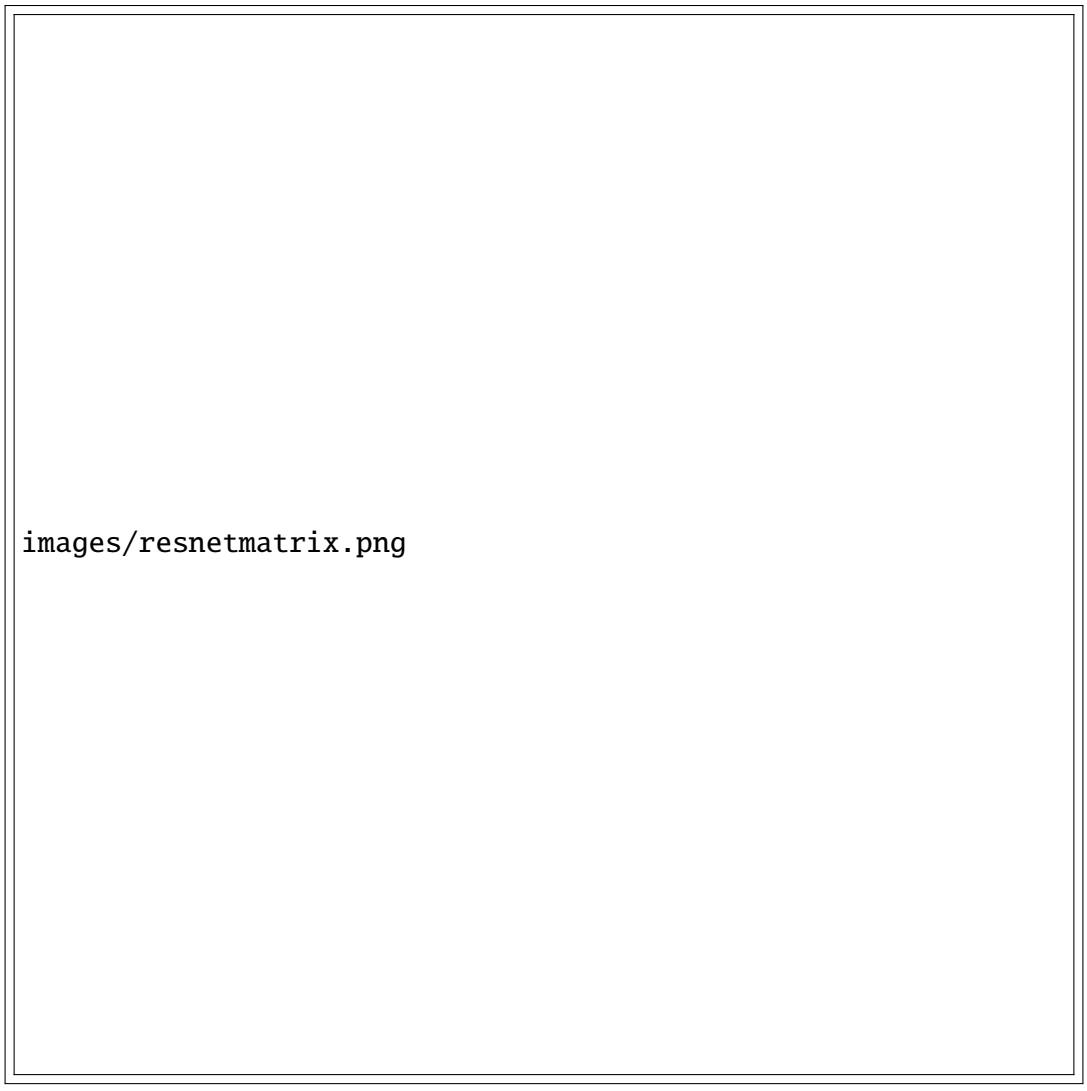

Figure 3: Confusion matrix for the ResNet50 classifier across all mushroom species. Most predictions lie on the diagonal, but the many off-diagonal entries show frequent confusion between visually similar species.

4.2 YOLOv8 Classification Pipeline

Data Preparation

The YOLOv8 pipeline reuses the same underlying images but reorganizes them into a directory structure suitable for Ultralytics classification:

- For each species, images are split into train, validation, and test sets using per-class proportions (approximately 70% / 15% / 15%).
- A helper function copies each image into a path of the form `yolov8_mushroom_dataset/split/species_n` where `split` is one of {`train`, `val`, `test`}.

This structure allows the Ultralytics library to automatically infer class labels from folder names and handle all splits consistently.

Model and Training Configuration

We use the YOLOv8 classification variant with pretrained weights:

- Model configuration: "yolov8s-cls.yaml" (small classification model).
- Pretrained weights: "yolov8s-cls.pt" (trained on ImageNet-like data).

Training is launched via the Ultralytics API:

```
model = YOLO("yolov8s-cls.yaml").load("yolov8s-cls.pt")
results = model.train(
    data="yolov8_mushroom_dataset",
    epochs=50,
    imgsz=...,
    hsv_v=0.4,
    flipud=0.2,
    degrees=15.0,
    translate=0.2,
    scale=0.8
)
```

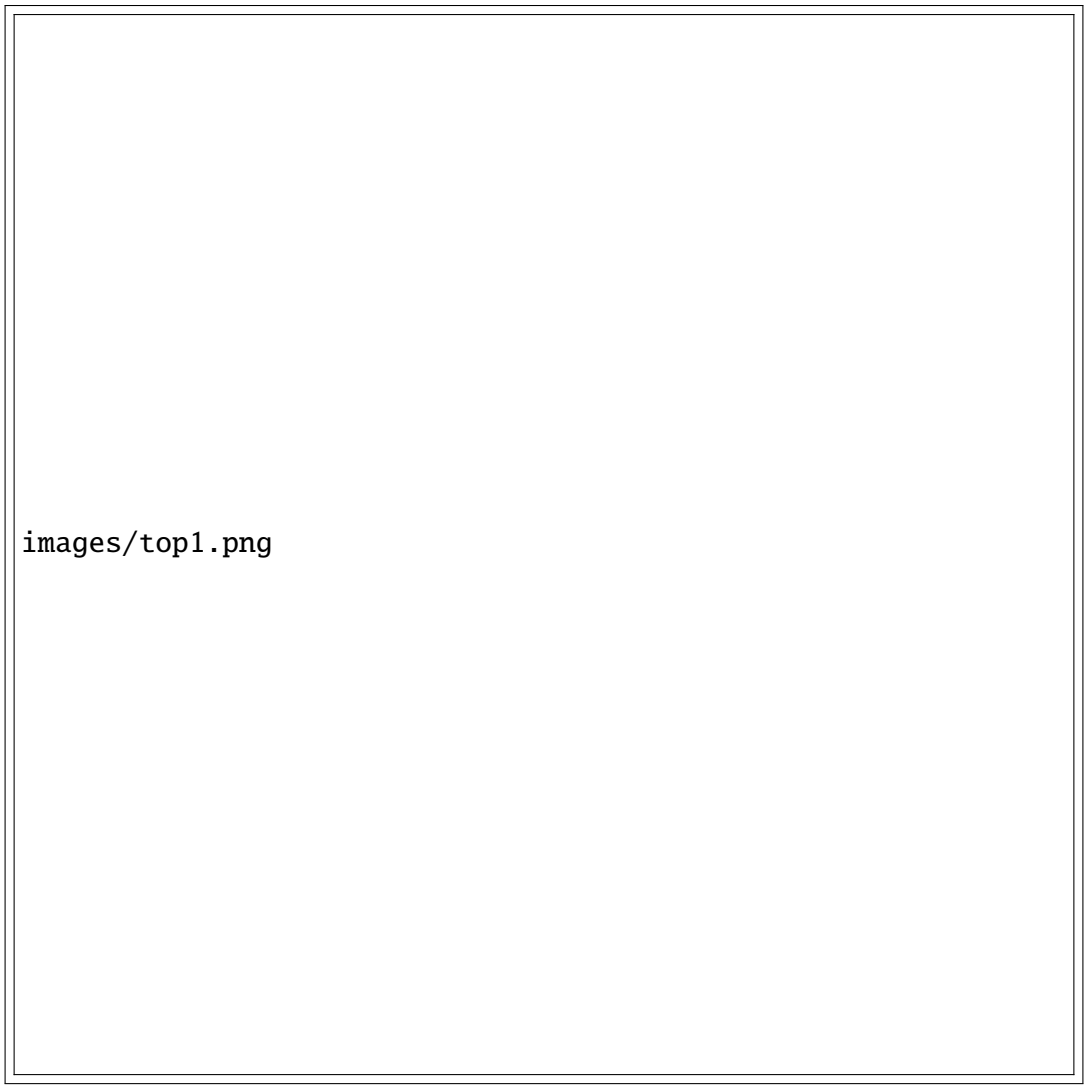

Figure 4: Training loss and validation top-1/top-5 accuracy curves for the YOLOv8 classifier. Loss decreases steadily while validation accuracy gradually improves over the 50 training epochs.

Here, YOLOv8 performs its own augmentations (brightness, flips, rotations, translations, and scaling) and handles batching and logging. After training, we evaluate on the held-out test set using:

```
val_results = model.val(
    data="yolov8_mushroom_dataset",
    batch=64,
    imgsz=...,
    split="test"
)
```

Ultralytics reports metrics such as top-1 accuracy and exports a confusion matrix image and training curves.

Performance Summary

The YOLOv8 classifier achieves test accuracy and F1-score in the same general range as the ResNet50 pipeline. While exact numbers vary with hyperparameters and random seeds, both models show:

- Meaningful learning above random chance.
- Persistent confusion among many visually similar species.

Figure 5: Validation accuracy for the YOLOv8 classifier over epochs. Accuracy rises quickly at the start of training and then plateaus around a moderate level, similar to the ResNet50 model.

The confusion matrix produced by YOLOv8 displays the same qualitative pattern observed for ResNet50: clustering of correct predictions for some species but widespread misclassification for others, especially those with small sample sizes or highly similar appearances.

4.3 Comparative Analysis

Comparing the two pipelines:

- **Architecture differences:** ResNet50 uses a classic residual CNN backbone with a custom classifier head, while YOLOv8 uses a modern, compact classification backbone with its own training regime and augmentations.
- **Performance:** Both reach moderate accuracy but fall well short of perfect species recognition.
- **Error patterns:** Both confusion matrices show that the hardest species tend to be those with few examples and those that are visually similar to other classes. Neither architecture resolves these ambiguities consistently.

From a research perspective, the key takeaway is that *two reasonably strong CNN-based architectures produce broadly similar performance and error patterns on this dataset*, which strengthens our conclusions about the limitations of image-only toxicity prediction.

5 Discussion & Conclusion

5.1 Discussion of Results

Our ResNet50 and YOLOv8 pipelines both achieve significant performance on multi-class mushroom species classification, but they also reveal important limitations:

- **Moderate accuracy only.** Overall test accuracies in the range of roughly 50–55% across 210 classes indicate that the models are learning some stable visual patterns, yet misclassification remains common.
- **Fine-grained difficulty.** Many species differ only subtly in color, texture, or shape. Even with transfer learning and augmentation, the models struggle to consistently distinguish such fine grained categories.
- **Sparse and imbalanced data.** Most species have limited training data (10–20 images), which is far from ideal for deep learning. Class weighting helps but cannot fully overcome the scarcity of examples.
- **Real-world variability.** Variation in lighting, background clutter, camera angle, and occlusions increases intra-class diversity and further complicates learning.

When we relate these findings back to the toxicity question, the implications are clear. Even if we had a mapping from species to toxicity labels, any toxicity prediction system based on these models would inherit their misclassifications. In practice, mistaking a poisonous species for a visually similar non-poisonous species (or vice versa) is exactly the type of error that must be avoided in a safety-critical application.

5.2 Hypothesis Evaluation

We began with the hypothesis that visual features in mushroom images—cap color and shape, texture, gills, and stem morphology, could be used by a CNN to accurately predict whether a mushroom is poisonous or non-poisonous. Because our dataset lacks toxicity labels, we instead evaluated CNNs on species recognition and interpreted the results as indirect evidence.

Our empirical findings show that:

- CNNs can indeed learn meaningful visual features and outperform random guessing by a wide margin on species classification.
- However, classification accuracy and F1-scores remain far from the "high reliability" threshold one would demand for life-critical toxicity predictions.
- Two different CNN architectures (ResNet50 and YOLOv8) yield similar levels of performance and error patterns, suggesting that the limitation is not specific to a single model.

Taken together, we conclude:

> **Within the scope of our dataset, model architectures, and experiments, visual features in mushroom images alone are not sufficient for a CNN to accurately and safely predict whether a mushroom is poisonous or non-poisonous.**

While the models show promising ability to recognize certain species, the overall error rate and fine grained ambiguities make the toxicity prediction hypothesis unsupported under the current conditions.

Future Work

Although our current results are not strong enough to recommend CNN-based toxicity prediction, they suggest several promising directions for future work:

- **Datasets with explicit toxicity labels.** Construct or obtain a curated dataset where each image is labeled not only by species but also by toxicity category (e.g., edible, poisonous, hallucinogenic). Training directly on these labels would allow a more precise test of the toxicity prediction hypothesis.
- **More images per class.** Collecting a larger number of images per species would improve generalization, reduce overfitting, and help the models learn subtle visual cues.
- **Metadata integration.** Incorporate non-visual features such as habitat, geographic location, and seasonality, which human experts commonly use alongside visual information.
- **Alternative architectures.** Explore other architectures such as EfficientNet, Vision Transformers, or ensemble models that combine multiple backbones or training regimes.
- **Explainability and feature analysis.** Use saliency maps, Grad-CAM, or other explainability tools to identify which parts of the mushroom (cap, gills, stem) the models rely on most, and compare these patterns to mycological knowledge.

These extensions would be necessary to revisit the toxicity prediction question with stronger evidence and a safer, more robust modeling setup.

References

[1] Daniil Onishchenko, *Mushrooms images classification 215*, Kaggle, accessed 2025. Available at: https://www.kaggle.com/datasets/daniilonishchenko/mushrooms-images-classificatio

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, *Deep Residual Learning for Image Recognition*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[3] Ultralytics, *YOLOv8: Real-Time Object Detection and Image Classification*, Ultralytics Documentation, accessed 2025. Available at: https://docs.ultralytics.com.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning*, MIT Press, 2016.