1. Introduction

During the autumn season, Finland's forests are abundant with various mushroom species. While mushrooms can serve as an excellent source of nutrition, accurate identification is essential to ensure safety, as some species are poisonous. To facilitate safe consumption, reliable methods for mushroom identification are required. One practical approach is to classify mushrooms based on photographic images. . In this study, we implemented a convolutional neural network (CNN) model with a random forest classifier to perform image-based mushroom classification.

Our report is structured as follows: In section 2, we define our classification problem in detail. The dataset and models are described in section 3 and in section 4 we present the result of our classification. In section 5, we discuss the results and in section 6. our use of AI. In section 7. are our appendices and in section 8. our references.

2. Problem formulation

We are trying to identify different mushroom species from photos of mushrooms. The goal is to input an image of a mushroom and the output should be the species of that mushroom. We started with a dataset that contains about one hundred thousand photos of 169 different species of mushroom. The dataset consists of the mushroom pictures and 3 .csv files that link the pictures to the mushroom species. We noticed that we had too many pictures compared to our resources, and decided to remove 100 mushroom species entirely, so in the end we had 69 species. For the initial training of the two different models, we also removed 50% of the pictures. The final model used approximately 45000 pictures.

The type of the data is categorial, since the mushrooms are divided into 69 species which are in a sense different categories. Our dataset is from kaggle and is compiled by Leonardo Cofone. (link to dataset)

This machine learning problem is a multi-class image classification problem. The images used as input are multidimensional, because each image consists of 224x224 pixels, and each pixel consists of 3 rgb values. This means that as a starting point, we have 224x224x3 = 150528 features per image, and each feature is an 8 bit RGB value that represents pixel intensity.

This project will use supervised learning, since we already know the correct output and are trying to teach the model the mappings on how to categorize the data correctly. The labels we will be using are the correct categories for each image, meaning that we have in total 169 possible labels. Each of the successfully classified images will have one label. The model that we are using will be the CNN, the convolutional neural network.

3. Methods

The dataset has approximately 45000 pictures of different mushrooms already labeled and divided in their corresponding folders. In addition, the data was already divided into training, validation and testing files, where 70% of the data is for training, 15% for validation and 15% for testing. We decided not to modify this division due to it being rather cumbersome and the distribution is about the standard that the industry uses. There are two csv files, for the training and validating, which link the respective images and image paths together.

3.1 Image Processing

Due to our datasets being initially rather large (12 gb) it was hard to process it without compressing it. We decided to load the dataset to a local machine and run a script (named format-images.py from our gitlab) to compress the pictures to the size of (224,224) which is also the preferred picture size of the CNN method. The sizing requires a 1x1 aspect ratio, so we decided to crop the images from the sides. In addition, we also used data augmentation to make the dataset more diverse. The technique we used was to flip the images and also change the brightness of the images. We randomly augmented 20% of our dataset. We then uploaded the compressed and modified dataset to google drive for easier access. (link to the google drive)

Our dataset had already labelled the pictures into the mushroom species, so we did not have to do any image labeling.

3.2 Chosen methods

Our first model was a Convolutional Neural Network (CNN) (LeCun, 2025), a deep learning model designed for image data. The CNN detects local patterns like edges, textures, and shapes through convolutional layers and combines them hierarchically to capture complex structures, with pooling layers reducing dimensionality and improving generalization. The CNN suits our project well, as it automatically extracts meaningful visual features, enabling accurate classification of mushroom species based on subtle patterns.

In the CNN, we used the ResNet50 model (Modi, 2025) as our main machine learning model. ResNet50 is a pretrained CNN that has been trained on over 1.2 million images across more than 1,000 categories. It operates on images sized 224x224 pixels, which matches the size of the images used in our dataset. By fine-tuning the pretrained ResNet50 model, we were able to adapt its learned features to recognize and classify different mushroom categories efficiently. We implemented this model with pytorch.

Our second model was Random Forest Classification (GeeksforGeeks, 2025), an ensemble method that builds multiple decision trees on random subsets of data and features and combines their outputs for robust predictions. Applied to the CNN-extracted image features, Random Forest captures complex, non-linear relationships while remaining interpretable, allowing us to evaluate a traditional machine learning approach for mushroom species classification.

3.3 Feature selection

In our project, the Convolutional Neural Network (CNN) does automatic feature extracting. As an image passes through the CNN, convolutional layers apply multiple filters to detect local patterns such as edges, textures, and shapes. These patterns are progressively combined in deeper layers, forming hierarchical features that capture increasingly complex structures of the image. Pooling layers further condense the information, focusing on the most important visual cues while reducing dimensionality and noise. The output of the CNN is a feature vector that encodes the visual characteristics necessary for distinguishing between mushroom species.

The random forest classifier uses these same features generated by the CNN.

3.4. Loss function

We chose to use the cross-entropy loss function for this project. It measures the difference between the estimated probability distribution q and the true distribution p.

$$H(p, q) = -\sum_{x \in \mathcal{X}} p(x) \log q(x).$$

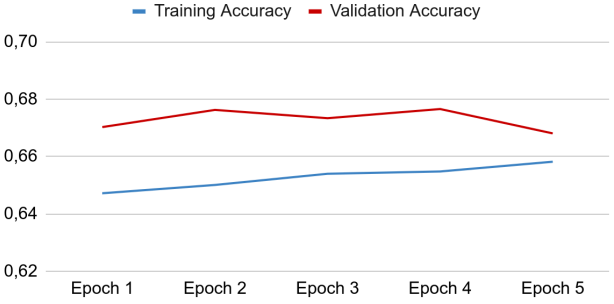
Cross-entropy is particularly suitable for multi-class classification problems as it penalizes the model more when it assigns a low probability for the correct mushroom class and it encourages the network to assign high probability to the true class. By minimizing this loss during training, the ResNet50 model gradually adjusts its weights to improve the accuracy of mushroom classification.

For the Random Forest classifier the concept of the loss function is different. Random Forests are ensemble-based decision tree models that do not rely on gradient-based optimization like neural networks. Instead, the model builds multiple decision trees using subsets of the features and samples, and makes predictions based on majority voting. In this project the Random Forest model used the features extracted from ResNet50, allowing it to leverage the rich visual representations for more accurate mushroom class predictions.

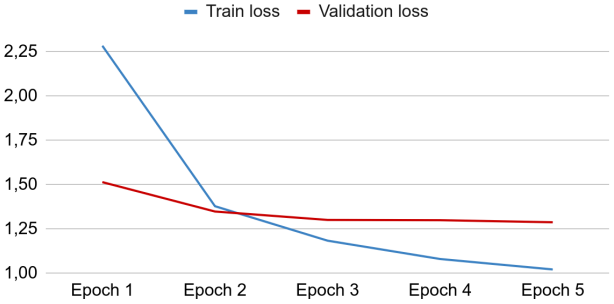
4. Results

For the best CNN model (Epoch 4), the average training accuracy was 0.6548 and the average training loss was 1.079 (Images 1 and 2). For the random forest, the average training accuracy was 0.9290 and training loss was 2.1897.

CNN Training and validation accuracy per Epoch (Image 1)



CNN training and validation loss per Epoch (Image 2)



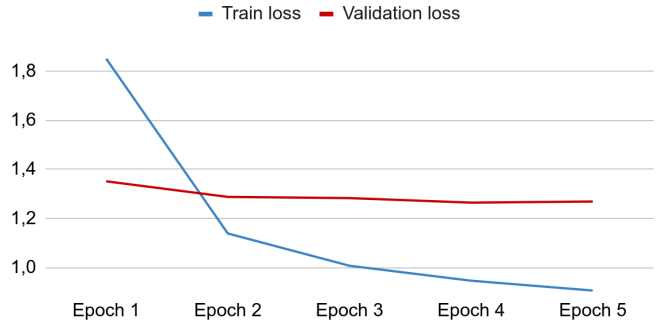
The average validation accuracy for the CNN was 0.6766 and the validation loss for the was 1.2984. (Images 1 and 2). For the random forest, the average validation accuracy was 0.01430 and the training loss was 3.9543.

Based on these results, the CNN performed significantly better with better accuracy both in training and validation and smaller training loss both in training and validation. Thus we decided to choose CNN as our final model. We trained our chosen CNN with all the picture data from the 69 mushroom species, and achieved the training loss of 0.9063 and training accuracy of 0.9648 from the best epoch (epoch 5) and the validation loss of 1.2687 and validation accuracy of 0.8375. (Images 3 and 4)

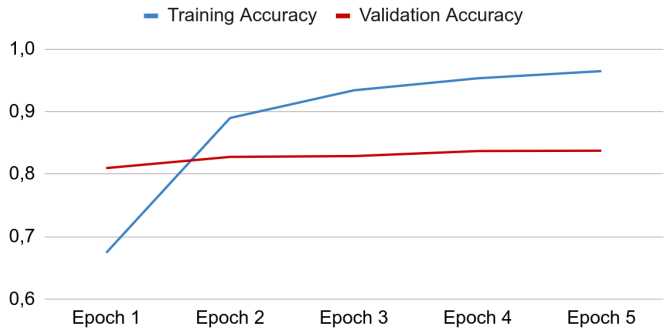
To assess the final CNN model's generalization to completely unseen data, we evaluated it on the test set, which had not been used during training or validation. The test set consisted of approximately 15% of the original dataset.

The CNN achieved an average test accuracy of 0.8124 and a test loss of 1.342. As expected, the test accuracy was slightly lower than the validation accuracy (0.8375), and the test loss slightly higher than the validation loss (1.2687). This small drop in performance indicates that the model generalizes reasonably well to new images, although there is still a minor gap that could be reduced with further improvements such as additional data augmentation, more training epochs, or fine-tuning on harder-to-classify species.

Final model (CNN) train and validation loss per epoch (Image 3)



Final model (CNN) train and validation accuracy per epoch (Image 4)



5. Conclusions

In this project, we found that it is possible to use machine learning to identify mushroom pictures. The CNN was found to be quite effective, with an accuracy of 0.8375 in identifying the pictures. This means that the model successfully identified over 80% of the mushroom pictures. The change over different epochs was just starting to reach a plateau, which means that we probably did enough epochs. Most likely overfitting did not yet happen, because the change over epochs had not started to worsen.

In our second model, we combined the ResNet-50 feature extractor with a Random Forest classifier. However, the implementation did not perform as well as expected, achieving only 14% accuracy on the validation set. We identified several factors that likely contributed to this outcome. Most importantly, we suspect that the Random Forest was accidentally trained on features extracted from images that the ResNet-50 model had already seen during its fine-tuning phase. Because the CNN was already familiar with these images, the extracted features became highly separable for the training data, leading the Random Forest to achieve very high training accuracy but poor generalization to unseen validation images. We believe that retraining the Random Forest on features extracted from a separate, unseen dataset would significantly improve its validation performance. Due to time constraints, we did not manage to do the retraining for this report.

6. Use of AI

In this project, AI was used to help with the coding of the models. The models that we were using became extremely difficult to implement, especially getting a model so fast that it could be run in a sensible amount of time. So AI was used to help make the code run faster and be more efficient with the computer's resources.

7. Appendices

Scripts for formatting and resizing the images

```
# This script converts images to the wanted size and format.
import os
from PIL import Image


input_folder = "Arkisto/merged_dataset"


output_folder = "dataset_resized"
os.makedirs(output_folder, exist_ok=True)


target_size = (224, 224)


for root, _, files in os.walk(input_folder):
   for file in files:
       if file.lower().endswith((".png", ".jpg", ".jpeg")):
           input_path = os.path.join(root, file)

           relative_path = os.path.relpath(root, input_folder)
           output_dir = os.path.join(output_folder, relative_path)
           os.makedirs(output_dir, exist_ok=True)

           output_path = os.path.join(output_dir, file)

           try:
               img = Image.open(input_path)

               img = img.convert("RGB")

               img = img.resize(target_size, Image.LANCZOS)

               img.save(output_path, "JPEG", quality=90)

           except Exception as e:
               print(f"Could not process {input_path}: {e}")

print("Resizing complete! All resized images saved in:", output_folder)
```

Scripts for creating data augmentation to the images

```
import os
from PIL import Image, ImageEnhance
import random


input_folder = "./merged_datasets_not_augmented"
output_folder = "./data/mushrooms_augmented"
os.makedirs(output_folder, exist_ok=True)

num_augments = 2


def augment_image(img):
    if random.random() > 0.5:
        img = img.transpose(Image.FLIP_LEFT_RIGHT)

    if random.random() > 0.5:
        img = img.transpose(Image.FLIP_TOP_BOTTOM)

    angle = random.randint(0, 1)
    if angle == 0:
        img = img.rotate(-90)
    elif angle == 0:
        img = img.rotate(-90)

    enhancer = ImageEnhance.Brightness(img)
    img = enhancer.enhance(random.uniform(0.7, 1.3))

    return img


for root, _, files in os.walk(input_folder):
    for file in files:
        if file.lower().endswith((".png", ".jpg", ".jpeg")):
            input_path = os.path.join(root, file)
            relative_path = os.path.relpath(root, input_folder)
            output_dir = os.path.join(output_folder, relative_path)
            os.makedirs(output_dir, exist_ok=True)

            img = Image.open(input_path).convert("RGB")

            img.save(os.path.join(output_dir, file))

            if random.random() < 0.2:
```

```
                img = Image.open(input_path).convert("RGB")
                aug_img = augment_image(img)
                aug_img.save(os.path.join(output_dir, file))

print("Data augmentation complete! Augmented images saved in:", output_folder)
```

Code for removing some mushroom classes from the .csv file

```
import pandas as pd
import os
import random
import shutil

# === CONFIG ===
train_csv_path = "train_original.csv"  # Your main dataset CSV
val_csv_path = "val_original.csv"  # Your validation CSV
output_train_csv = "train.csv"
output_val_csv = "val.csv"
delete_count = 100  # Number of unique species to remove
trash_dir = "deleted_images"  # Folder to move deleted images into

# === PREPARE OUTPUT DIR ===
os.makedirs(trash_dir, exist_ok=True)

# === LOAD CSVs ===
train_df = pd.read_csv(train_csv_path)
val_df = pd.read_csv(val_csv_path)

required_cols = {"label", "image_path"}
if not required_cols.issubset(train_df.columns) or not required_cols.issubset(
    val_df.columns
):
    raise ValueError("Both CSVs must contain 'label' and 'image_path' columns")

# === CHOOSE SPECIES TO DELETE ===
unique_species = train_df["label"].unique()

if len(unique_species) <= delete_count:
    raise ValueError(
                f"Only {len(unique_species)} unique species found — can't delete
{delete_count}"
    )

species_to_delete = random.sample(list(unique_species), delete_count)
```

```
print(f"\nDeleting {len(species_to_delete)} species:")
for s in species_to_delete:
    print(f" - {s}")


# === DELETE FILES (TRAIN + VAL) ===
def move_images(df, species_list):
    moved = 0
    for _, row in df.iterrows():
        if row["label"] in species_list:
            img_path = row["image_path"]
            if os.path.exists(img_path):
                # Move image to trash directory, preserving folder structure
                relative_path = os.path.relpath(
                    img_path, start=os.path.commonpath([img_path, trash_dir])
                )
                dest_path = os.path.join(trash_dir, os.path.basename(img_path))
                try:
                    shutil.move(img_path, dest_path)
                    moved += 1
                except Exception as e:
                    print(f"Error moving {img_path}: {e}")
    return moved


deleted_train = move_images(train_df, species_to_delete)
deleted_val = move_images(val_df, species_to_delete)
print(f"\nMoved {deleted_train} train and {deleted_val} val images to
'{trash_dir}'.")

# === FILTER OUT DELETED SPECIES ===
train_filtered = train_df[~train_df["label"].isin(species_to_delete)]
val_filtered = val_df[~val_df["label"].isin(species_to_delete)]

# === SAVE FILTERED CSVs ===
train_filtered.to_csv(output_train_csv, index=False)
val_filtered.to_csv(output_val_csv, index=False)

print(f"\nFiltered train CSV saved to: {output_train_csv}")
print(f"Filtered val CSV saved to:   {output_val_csv}")
print("\nDone!")
```

Code for training the ResNet50 model

```
import os
import time
import pandas as pd
from PIL import Image
import matplotlib.pyplot as plt
from tqdm import tqdm

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset

from torchvision import models
from albumentations import (
    Compose,
    RandomResizedCrop,
    HorizontalFlip,
    VerticalFlip,
    Affine,
    ColorJitter,
    GaussianBlur,
    Normalize,
)
from albumentations.pytorch import ToTensorV2



# -------------------------
# Dataset
# -------------------------
class CSVImageDataset(Dataset):
    def __init__(self, csv_file, root_dir, transform=None, class_to_idx=None):
        self.root_dir = root_dir
        self.transform = transform
        annotations = pd.read_csv(csv_file)

        if (
            "image_path" not in annotations.columns
            or "label" not in annotations.columns
        ):
            raise ValueError("CSV must contain columns 'image_path' and 'label'")

        annotations["full_path"] = annotations["image_path"].apply(
            lambda p: os.path.join(self.root_dir, str(p).lstrip("/"))
```

```
        )

        missing_mask = ~annotations["full_path"].apply(os.path.exists)
        n_missing = missing_mask.sum()
        if n_missing > 0:
            print(f"WARNING: {n_missing} missing images ignored.")
            annotations = annotations[~missing_mask].reset_index(drop=True)

        self.annotations = annotations
        if class_to_idx is None:
            self.classes = sorted(annotations["label"].unique())
            self.class_to_idx = {cls: i for i, cls in enumerate(self.classes)}
        else:
            self.class_to_idx = class_to_idx
            self.classes = sorted(class_to_idx.keys())

                                              self.annotations["label_idx"]       =
self.annotations["label"].map(self.class_to_idx)
        if self.annotations["label_idx"].isnull().any():
            raise ValueError("Some labels not found in class_to_idx mapping.")

    def __len__(self):
        return len(self.annotations)

    def __getitem__(self, idx):
        row = self.annotations.iloc[idx]
        image = Image.open(row["full_path"]).convert("RGB")
        image = np.array(image)
        if self.transform:
            image = self.transform(image=image)["image"]
        return image, int(row["label_idx"])



# -------------------------
# Training Function
# -------------------------
def train_model(
    model,
    criterion,
    optimizer,
    scheduler,
    dataloaders,
    device,
    num_epochs=25,
    checkpoint_path="checkpoint.pth",
```

```
):
    since = time.time()
    best_acc = 0.0
    history = {"train_loss": [], "val_loss": [], "train_acc": [], "val_acc": []}

    for epoch in range(num_epochs):
        print(f"\nEpoch {epoch + 1}/{num_epochs}\n" + "-" * 20)

        for phase in ["train", "val"]:
            model.train() if phase == "train" else model.eval()
            running_loss, running_corrects = 0.0, 0
            dataset_size = len(dataloaders[phase].dataset)

            loop = tqdm(dataloaders[phase], desc=f"{phase}", leave=False)
            for inputs, labels in loop:
                inputs, labels = inputs.to(device), labels.to(device)
                optimizer.zero_grad()

                with torch.set_grad_enabled(phase == "train"):
                    outputs = model(inputs)
                    loss = criterion(outputs, labels)
                    _, preds = torch.max(outputs, 1)

                    if phase == "train":
                        loss.backward()
                        optimizer.step()

                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels).item()

            if phase == "train":
                scheduler.step()

            epoch_loss = running_loss / dataset_size
            epoch_acc = running_corrects / dataset_size
            history[f"{phase}_loss"].append(epoch_loss)
            history[f"{phase}_acc"].append(epoch_acc)
                        print(f"{phase.capitalize()}  Loss:  {epoch_loss:.4f}  Acc:
{epoch_acc:.4f}")

            if phase == "val" and epoch_acc > best_acc:
                best_acc = epoch_acc
                torch.save(model.state_dict(), "best_mushroom_model.pth")
                print("Saved new best model!")
```

```
            torch.save(
                {
                    "epoch": epoch + 1,
                    "model_state_dict": model.state_dict(),
                    "optimizer_state_dict": optimizer.state_dict(),
                    "best_acc": best_acc,
                },
                checkpoint_path,
            )

    total_time = time.time() - since
    print(f"\nTraining complete in {int(total_time // 60)}m {int(total_time % 60)}s")
    print(f"Best validation accuracy: {best_acc:.4f}")
    model.load_state_dict(torch.load("best_mushroom_model.pth"))
    return model, history


# -------------------------
# Plot History
# -------------------------
def plot_history(history, out_path="training_history.png"):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))
    ax1.plot(history["train_loss"], label="Train Loss")
    ax1.plot(history["val_loss"], label="Val Loss")
    ax1.set_title("Loss")
    ax1.legend()
    ax2.plot(history["train_acc"], label="Train Acc")
    ax2.plot(history["val_acc"], label="Val Acc")
    ax2.set_title("Accuracy")
    ax2.legend()
    plt.tight_layout()
    plt.savefig(out_path)
    plt.show()


# -------------------------
# Main
# -------------------------
if __name__ == "__main__":
    import numpy as np

    DATA_ROOT = "."
    TRAIN_CSV, VAL_CSV = "train_reduced_final.csv", "val_reduced_final.csv"
    NUM_EPOCHS, BATCH_SIZE = 30, 1024
```

```
    CHECKPOINT_PATH = "checkpoint.pth"

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f"Using device: {device}")

    # ------------------------
    # Transforms (Albumentations)
    # ------------------------
    train_transform = Compose(
        [
            RandomResizedCrop(size=(224, 224), scale=(0.8, 1.0)),
            HorizontalFlip(p=0.5),
            VerticalFlip(p=0.3),
            Affine(
                  scale=(0.9, 1.1), translate_percent=(0.1, 0.1), rotate=(-15, 15),
p=0.5
            ),
                 ColorJitter(brightness=0.3, contrast=0.3, saturation=0.3, hue=0.2,
p=0.5),
            GaussianBlur(p=0.2),
            Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
            ToTensorV2(),
        ]
    )

    val_transform = Compose(
              [Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
ToTensorV2()]
    )

    # ------------------------
    # Datasets
    # ------------------------
    train_dataset = CSVImageDataset(TRAIN_CSV, DATA_ROOT, transform=train_transform)
    class_to_idx = train_dataset.class_to_idx
    val_dataset = CSVImageDataset(
        VAL_CSV, DATA_ROOT, transform=val_transform, class_to_idx=class_to_idx
    )

    print(
            f"Found {len(class_to_idx)} classes. Train: {len(train_dataset)}, Val:
{len(val_dataset)} images."
    )

    # ------------------------
```

```python
# DataLoaders (High RAM Setup)
# -------------------------
dataloaders = {
    "train": DataLoader(
        train_dataset,
        batch_size=BATCH_SIZE,
        shuffle=True,
        num_workers=os.cpu_count() // 2,
        prefetch_factor=4,
        persistent_workers=True,
    ),
    "val": DataLoader(
        val_dataset,
        batch_size=BATCH_SIZE,
        shuffle=False,
        num_workers=os.cpu_count() // 2,
        prefetch_factor=4,
        persistent_workers=True,
    ),
}

# -------------------------
# Model (ResNet50 fine-tuned)
# -------------------------
model = models.resnet50(weights=models.ResNet50_Weights.IMAGENET1K_V1)
for name, param in model.named_parameters():
    if "layer4" in name or "fc" in name:
        param.requires_grad = True
    else:
        param.requires_grad = False

num_features = model.fc.in_features
model.fc = nn.Sequential(
    nn.Dropout(0.3), nn.Linear(num_features, len(class_to_idx))
)
model = model.to(device)

try:
    model = torch.compile(model, mode="reduce-overhead")
except Exception as e:
    print(f"torch.compile not available: {e}")

criterion = nn.CrossEntropyLoss(label_smoothing=0.1)
optimizer = optim.AdamW(
    filter(lambda p: p.requires_grad, model.parameters()),
```

```
        lr=1e-4,
        weight_decay=1e-4,
    )
    scheduler = optim.lr_scheduler.CosineAnnealingLR(
        optimizer, T_max=NUM_EPOCHS, eta_min=1e-6
    )

    # -------------------------
    # Train Model
    # -------------------------
    trained_model, history = train_model(
        model,
        criterion,
        optimizer,
        scheduler,
        dataloaders,
        device,
        num_epochs=NUM_EPOCHS,
        checkpoint_path=CHECKPOINT_PATH,
    )

    plot_history(history)
    print("Training finished. Best model saved to 'best_mushroom_model.pth'")
```

Code for training the Random Forest classifier

```
import torch
from torch.utils.data import DataLoader
from torchvision import transforms, models
from PIL import Image
import pandas as pd
import os
import numpy as np
from tqdm import tqdm
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score, classification_report
import joblib



# ----------------------------
# CSVImageDataset class
# ----------------------------
```

```
class CSVImageDataset(torch.utils.data.Dataset):
    def __init__(self, csv_file, root_dir, transform=None, class_to_idx=None):
        self.annotations = pd.read_csv(csv_file)
        self.root_dir = root_dir
        self.transform = transform

        if class_to_idx is None:
            self.classes = sorted(self.annotations["label"].unique())
                        self.class_to_idx = {cls_name: i for i, cls_name in
enumerate(self.classes)}
        else:
            self.class_to_idx = class_to_idx
            self.classes = list(class_to_idx.keys())

    def __len__(self):
        return len(self.annotations)

    def __getitem__(self, idx):
        row = self.annotations.iloc[idx]
        path_from_csv = row["image_path"].lstrip("/")
        img_path = os.path.join(self.root_dir, path_from_csv)

        try:
            image = Image.open(img_path).convert("RGB")
        except FileNotFoundError:
            print(f"[WARNING] Missing file: {img_path}")
            return torch.zeros((3, 224, 224)), -1

        label = self.class_to_idx[row["label"]]
        if self.transform:
            image = self.transform(image)
        return image, label


# ------------------------------
# Configuration
# ------------------------------
train_csv = "train_reduced_final.csv"
test_csv = "val_reduced_final.csv"
root_dir = ""
batch_size = 32
num_workers = 2
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)
```

```
# ------------------------------
# Transforms
# ------------------------------
data_transforms = transforms.Compose(
    [
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
    ]
)



# ------------------------------
# Load datasets
# ------------------------------
train_dataset = CSVImageDataset(train_csv, root_dir, transform=data_transforms)
test_dataset = CSVImageDataset(
    test_csv,
    root_dir,
    transform=data_transforms,
    class_to_idx=train_dataset.class_to_idx,
)

train_loader = DataLoader(
    train_dataset, batch_size=batch_size, shuffle=False, num_workers=num_workers
)
test_loader = DataLoader(
    test_dataset, batch_size=batch_size, shuffle=False, num_workers=num_workers
)



# ------------------------------
# Load pretrained CNN (.pth)
# ------------------------------
cnn_path = "mushroom_model.pth"

if os.path.exists(cnn_path):
    print(f"Loading pretrained CNN from '{cnn_path}'...")
    resnet = models.resnet18(weights=None)
    resnet.fc = torch.nn.Identity()
    checkpoint = torch.load(cnn_path, map_location=device)
    resnet.load_state_dict(checkpoint, strict=False)
else:
    print("No .pth model found, using ImageNet pretrained ResNet18...")
    resnet = models.resnet18(weights="IMAGENET1K_V1")
```

```
    resnet.fc = torch.nn.Identity()

resnet = resnet.to(device)
resnet.eval()



# ------------------------------
# Feature extraction function
# ------------------------------
def extract_features(dataloader, save_features_path, save_labels_path):
    features_list, labels_list = [], []
    total = 0

    with torch.no_grad():
        for imgs, labels in dataloader:
            imgs = imgs.to(device)
            feats = resnet(imgs).cpu().numpy()
            features_list.append(feats)
            labels_list.append(labels.numpy())
            total += imgs.size(0)
            if total % 1000 < batch_size:
                print(f"Extracted {total} features...")

    features = np.concatenate(features_list)
    labels = np.concatenate(labels_list)
    np.save(save_features_path, features)
    np.save(save_labels_path, labels)
    print(f"Done extracting {total} features — saved to {save_features_path}")
    return features, labels



# ------------------------------
# Extract or load cached features
# ------------------------------
if os.path.exists("train_features.npy") and os.path.exists("train_labels.npy"):
    print("Loading cached features...")
    train_features = np.load("train_features.npy")
    train_labels = np.load("train_labels.npy")
    test_features = np.load("test_features.npy")
    test_labels = np.load("test_labels.npy")
else:
    train_features, train_labels = extract_features(
        train_loader, "train_features.npy", "train_labels.npy"
    )
    test_features, test_labels = extract_features(
```

```
        test_loader, "test_features.npy", "test_labels.npy"
    )


# ------------------------------
# Optional: PCA for speed & performance
# ------------------------------
print("Running PCA (dimensionality reduction)...")
pca_model_path = "pca_model.joblib"

if os.path.exists(pca_model_path):
    pca = joblib.load(pca_model_path)
    print("Loaded existing PCA model.")
else:
    pca = PCA(n_components=100)
    pca.fit(train_features)
    joblib.dump(pca, pca_model_path)
    print("Trained and saved PCA model.")

train_pca = pca.transform(train_features)
test_pca = pca.transform(test_features)


# ------------------------------
# Train or load Random Forest
# ------------------------------
rf_model_path = "rf_model_2.joblib"

if os.path.exists(rf_model_path):
    print(f"Loading existing Random Forest model from '{rf_model_path}'...")
    rf = joblib.load(rf_model_path)
else:
    print("No existing Random Forest found — training new model...")
        rf  =  RandomForestClassifier(n_estimators=200,  n_jobs=-1,  random_state=42,
verbose=1)
    rf.fit(train_pca, train_labels)
    joblib.dump(rf, rf_model_path)
    print(f"Saved trained Random Forest to '{rf_model_path}'")


# ------------------------------
# Evaluate
# ------------------------------
print("Evaluating Random Forest model...")
preds = rf.predict(test_pca)
```

```
acc = accuracy_score(test_labels, preds)
print(f"\nTest Accuracy: {acc:.4f}")
print(
   "\nClassification Report:\n",
   classification_report(test_labels, preds, target_names=train_dataset.classes),
)
```

8. References:

GeeksforGeeks. “Random Forest Algorithm in Machine Learning.” *GeeksforGeeks*, 1 September 2025, https://www.geeksforgeeks.org/machine-learning/random-forest-algorithm-in-machine-learning/. Accessed 8 October 2025.

LeCun, Yann. “Convolutional Neural Network (CNN) in Machine Learning.” *GeeksforGeeks*, 23 July 2025, https://www.geeksforgeeks.org/deep-learning/convolutional-neural-network-cnn-in-machine-learning/. Accessed 19 September 2025.

Modi, Rohit. “ResNet — Understand and Implement from scratch | by Rohit Modi | Analytics Vidhya.” *Medium*, 1 December 2021, https://medium.com/analytics-vidhya/resnet-understand-and-implement-from-scratch-d0eb9725e0db. Accessed 19 September 2025.