OsloMet- Oslo Metropolitan University

Department of Computer Science

Oslo Norway

ACIT4630-1 24V Advanced Machine Learning and Deep Learning

May 2024

Optimizing Mushroom Classification: Leveraging Transfer Learning with EfficientNetB0 Architecture

Candidate number 107, 108 , 119

Contents

0.1 Abstract

Classifying mushrooms is an important taks with many applications in ecological studies, public health, and the culinary arts. In this work, we provide a novel method for improving the categorization of mushrooms into poisonous or edible. Approach of transfer learning was utlized and EfficientNetB0 architecture was choosen. We acquire datasets, perform thorough preprocessing, construct models, and optimize hyperparameters as part of our technique. By using transfer learning, we improved the pre-trained EfficientNetB0 model's classification accuracy by methodically fine tuning the hyper-parameter for optimum result. We found an ideal model configuration through repeated trying and testing, which greatly enhanced classification performance, especially in properly recognizing dangerous mushrooms.

Keywords: EfficientNetB0; convolutional neural network; deep learning; mushroom;

0.2 Introduction

The categorization of mushrooms, specifically the differentiation between toxic and non-toxic types, is an essential undertaking in multiple fields such as mycology, food safety, and medical science. Mushroom classification is particularly challenging as there are millions of different types of mushroom [**?**]. Toxic species should only be identified accurately to avoid potential harm from ingestion. Recent developments in machine learning methods have opened up exciting new possibilities for this classification process' automation.

Deep learning has widespread applications across various domains like healthcare, finance, autonomous vehicles, and natural language processing[**?**].Several studies have delved into the application of machine learning algorithms and deep learning architectures for mushroom classification. (Tarawneh et al., 2022) explored the efficacy of Decision Trees, Random Forests, and Support Vector Machines in categorizing mushrooms based on their features[**?**]. (Zhang et al., 2022) applied the CNN architecture to extract features from images and classify them into poisonous or edible. The paper uses only few species which are divided into poisonous and edible mushroom.

Moreover, significant advancements have been made in image classification, with the emergence of robust models trained on extensive datasets capable of recognizing diverse patterns within images. Transfer learning is a method which involves leveraging knowledge from a pre-trained model on a source task to improve performance on a target task[**?**]. Leveraging these pre-trained models on new datasets is facilitated by their comprehensive understanding of various patterns. This simplifies the process of training on novel images, as the pre-trained networks already possess rich information about different patterns. Through transfer learning, wherein pre-trained models are fine-tuned for new tasks, notable progress has been demonstrated in augmenting the diagnostic capabilities of these models. Tan and Le (2020) presented a ground-breaking study that introduced the EfficientNet design, which uses compound scaling to transform the field[**?**]. This novel methodology outperformed conventional scaling techniques, producing previously unheard-of efficiency-accuracy trade-offs. Among these, the EfficientNetB0 architecture is particularly noteworthy for being the best option for transfer learning because of its well-thought-out architecture and outstanding results in a variety of image classification tasks.

(Ketwongsa et al., 2022) investigated the potential of Convolutional Neural Networks (CNNs) for mushroom classification, leveraging image processing techniques to extract discriminative features from mushroom images. The paper also use the concept of transfer learning. It uses alex net as a base model and try to fine tune it for the mushroom classification task[**?**]. The paper use five species of mushroom found in Thailand classify it into poisonous and non poisonous and make machine learning model using transfer learning approach.

This essay aims to explore the optimization of mushroom classification through the integration of transfer learning with the EfficientNetB0 architecture. EfficientNetB0 being choosen, as it has the state of the art result, it excels at image recognition at the same time being efficient. By leveraging the pre-trained weights of EfficientNetB0 on large-scale image datasets, we seek to enhance the accuracy and efficiency of mushroom classification models. Drawing insights from the aforementioned studies, we aim to demonstrate the effectiveness of this approach

in distinguishing between poisonous and non-poisonous mushrooms, thereby contributing to advancements in food safety and mycology.

0.3 Methodology

Our journey to develop a robust mushroom classification system began with a comprehensive search for a dataset that would serve as the foundation of our machine learning project. We conducted extensive research, leveraging academic repositories, online datasets, and domain-specific forums to identify a suitable corpus of mushroom images. This process involved a meticulous examination of each dataset's composition, quality, and relevance to our research objectives.

After thorough deliberation and consultation with domain experts, we settled on a dataset comprising approximately 100,000 images sourced from diverse sources such as online repositories, scientific publications, and citizen science initiatives. On closer examination, however, we encountered several challenges inherent in the selected data set. Among these challenges, was the presence of multiple classes, which include poisonous, edible, conditionally edible, and deadly mushrooms, which complicated the classification task.

Furthermore, the dataset suffered from significant class imbalance, with the 'conditionally edible' class disproportionately represented, posing a potential bias in model training.

To mitigate these issues and streamline our research focus, we made the strategic decision to narrow our scope to the binary classification of mushrooms as either 'poisonous' or 'edible'. This decision not only aligned with the primary objectives of our study but also facilitated a more balanced and focused analysis. With the dataset curated and class imbalance addressed through careful selection, our attention turned to preprocessing the images to uniformity and compatibility with our chosen machine learning model.

The dataset contains different species of poisonous and non poisonous class. There were numerous challanges like varying pattern, color, shape, background. Many of them looks anything else but not mushroom.

(a) Mushroom 1 (b) Mushroom 2 (c) Mushroom 3 (d) Mushroom 4

(e) Mushroom 5 (f) Mushroom 6 (g) Mushroom 7 (h) Mushroom 8

Figure 1: Different kinds of poisonous mushrooms

The initial inspection revealed inconsistencies in image sizes, necessitating resizing to a standardized format of 224x224 pixels. Additionally, we identified discrepancies in image formats, with some images in JPEG, JPG, and PNG formats. Of particular concern was the presence of PNG images, which introduced an additional alpha channel representing transparency, thereby deviating from the RGB color space used by the majority of the dataset. To rectify this discrepancy and ensure consistency in input data, we systematically converted all PNG images to the JPEG format, thus harmonizing the color channels across the entire dataset. Moreover, during the preprocessing stage, we encountered a subset of images that were corrupted or of insufficient quality for meaningful analysis. To maintain data integrity and prevent noise from influencing model performance, these corrupted images were systematically identified and removed from the dataset. This meticulous approach to data preprocessing was essential to ensure the reliability and robustness of our machine learning pipeline.

After carefully preparing the dataset, our next crucial step was model selection and architecture design. Recognizing the importance of utilizing state-of-the-art techniques for image classification, we conducted a comprehensive survey of modern models and architectures. While we first assessed the ResNet model, which has shown success in various image classification tasks, our preliminary experiments revealed suboptimal performance on our mushroom classification task. In response, we shifted our focus to EfficientNetB0, a recent and highly efficient convolutional neural network (CNN) architecture known for its superior performance on image classification.

With the EfficientNetB0 architecture selected as our backbone, we proceeded to fine-tune the model using transfer learning—a technique that leverages pre-trained models on large-scale datasets to accelerate learning on domain-specific tasks with limited labeled data. By transferring knowledge from the pre-trained EfficientNetB0 model, which was trained on ImageNet—a vast dataset comprising millions of labeled images spanning thousands of classes—we aimed to expedite the training process and enhance the model's ability to generalize across diverse mushroom images.

0.4 Proposed architecture

Our mushroom classification model utilizes the EfficientNetB0 architecture, renowned for its efficiency and performance in image classification tasks. Fine-tuning the pre-trained EfficientNetB0 model on our dataset allows it to adapt its parameters to the specific task of mushroom classification while leveraging knowledge learned from large-scale image datasets. EfficientNetB0 is a unique convolutional neural network architecture because it prioritizes computational efficiency above performance to get cutting-edge results. To achieve that state of the art result, it proposes a few new ideas:

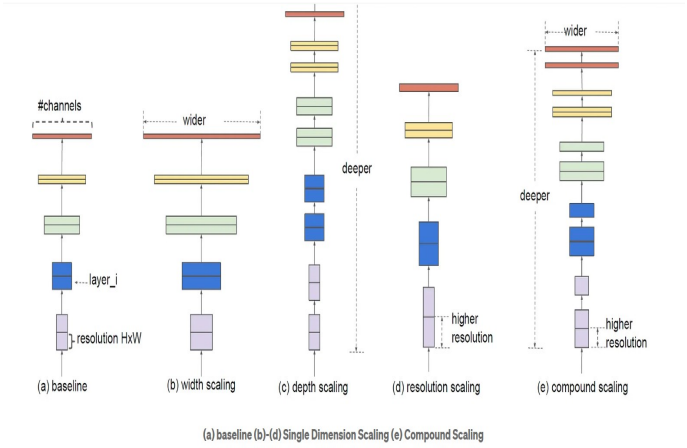


Figure 2: Figure 3 from Tan and Le (2020) illustrates the working of EfficientNetBO

1. **Compound Scaling:** By uniformly scaling the model's width, depth, and resolution with a set ratio, this novel approach guarantees a balanced adjustment of these dimensions. With this approach, the model's performance is maximized without incurring exponentially higher computing costs[**?**].

2. **Depthwise Separable Convolution:** EfficientNetB0 uses fewer computations and parameters than typical convolutional layers since it uses depthwise separable convolutional layers. This simplified architecture maintains representational capability while improving computing efficiency[**?**].

3. **Inverted Residual Blocks:** EfficientNetB0 integrates inverted residual blocks with linear bottlenecks, drawing inspiration from MobileNetV2. These building components enable effective information transfer across the network, reducing computing burden and preserving high-quality feature representation[**?**].

4. **Efficient Channel Attention (ECA):** By recalibrating channel-wise answers, ECA provides a lightweight technique for channel attention, improving feature representation. This addition increases the efficiency of the model by supporting its performance without dramatically increasing computational expenses[**?**].

5. **Swish Activation Function:** Using this activation function, which has smoother behavior than more conventional activation functions like ReLU, allows EfficientNetB0 to operate more efficiently. This decision preserves computational economy while improving performance on a variety of jobs[**?**].

6. **Model Scaling:** The compound coefficient (ϕ) was introduced to allow for scalable model design. This means that practitioners can adjust the size of the model according to the computational resources that are available and the performance levels that they want[**?**].

EfficientNet architecture consists of seven blocks which are shown in different colours. The basic building block of EfficientNet-B0 is a mobile inverted bottleneck convolution (MBConv), while each MBConv block is shown with the corresponding kernel filter size [**?**]. The figure below shows the different block:

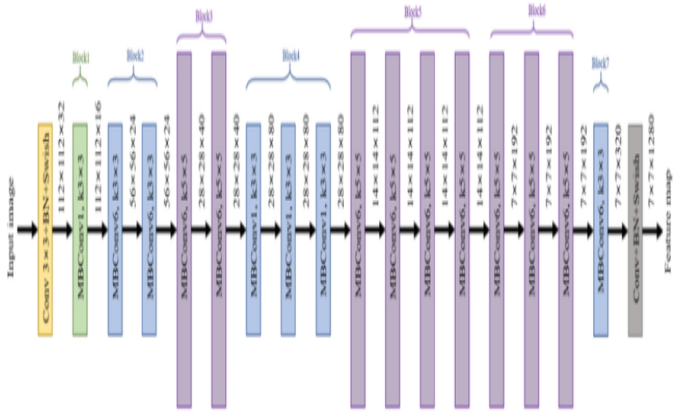


Figure 3: Figure 3 from Zhou et al. (2022) illustrates the architecture of EfficientNetBO

The model comprises convolutional layers followed by depthwise separable convolutions. We modify this EfficientNetB0 by adding few layers and a classification layer. Global average pooling was used to reduce the parameter size. This layer not just reduce parameter also preserve the most important features by taking out the average. On top of that dense layer with ReLU activation was used further enhancing the model's ability to learn complex pattern in images, dropout regularization was used ensuring that the model could be generalized and does better on test set rather than just doing good only on training set. And a classification layer with sigmoid activation was used for binary classification. This architecture enables the development of a robust classification model capable of accurately distinguishing between edible and poisonous mushrooms.

0.5 Experiment

Our experimental journey encompassed a meticulous exploration of various configurations, hyperparameters, and optimization strategies to refine our mushroom classification model and maximize its performance. Central to our experimentation was the systematic tuning of hyperparameters, a process that involved iteratively adjusting key variables to identify the optimal settings for our model.

Table 1: Tested with various hyperparameters

Optimizers	Learning rates	Dropout rates	Layers of base model removed
RMSprop,Adam	0.1–0.0000001	0.1–0.9	-1 up to -7

One of the primary hyperparameters under scrutiny was the learning rate—a critical parameter governing the magnitude of updates to the model's weights during training. Recognizing the profound impact of learning rate on model convergence and performance, we conducted a series of experiments spanning a wide range of values, from 0.1 to 0.0001, to elucidate its effects on training dynamics and final classification accuracy.

In parallel, we explored the effectiveness of different optimization algorithms in guiding the model's weight updates towards the optimal solution. Two prominent optimizers, RMSprop and Adam, were selected for evaluation based on their widespread adoption and proven effectiveness in training deep neural networks. Through rigorous experimentation, we sought to discern nuanced differences in optimization behavior and performance between these algorithms, thereby informing our choice of optimizer for subsequent training iterations.

Beyond hyperparameter tuning, our experimentation encompassed the systematic exploration of model architectures and configuration settings to unlock the full potential of our chosen backbone, the EfficientNetB0 architecture. Recognizing that the depth and complexity of neural network architectures can significantly influence model performance, we embarked on a journey of model pruning, systematically removing layers from the pre-trained EfficientNetB0 model to tailor its architecture to our specific classification task. This process involved iteratively removing different numbers of layers, ranging from one to seven, and evaluating the resulting architectures' performance on our validation dataset.

To further optimize the computational efficiency and generalization capacity of our model, we employed regularization techniques such as dropout—a popular method for mitigating overfitting by randomly deactivating neurons during training. By varying the dropout rate from 0.1 to 0.9, we explored its impact on model performance and stability, seeking to strike a balance between regularization strength and preservation of valuable information encoded in the network weights.

Throughout the experimentation phase, model performance was meticulously evaluated using a comprehensive suite of evaluation metrics, including accuracy, precision, recall, and F1-score. These metrics provide valuable insights into the model's ability to correctly classify mushrooms as either 'poisonous' or 'edible', while also shedding light on potential areas for improvement and refinement.

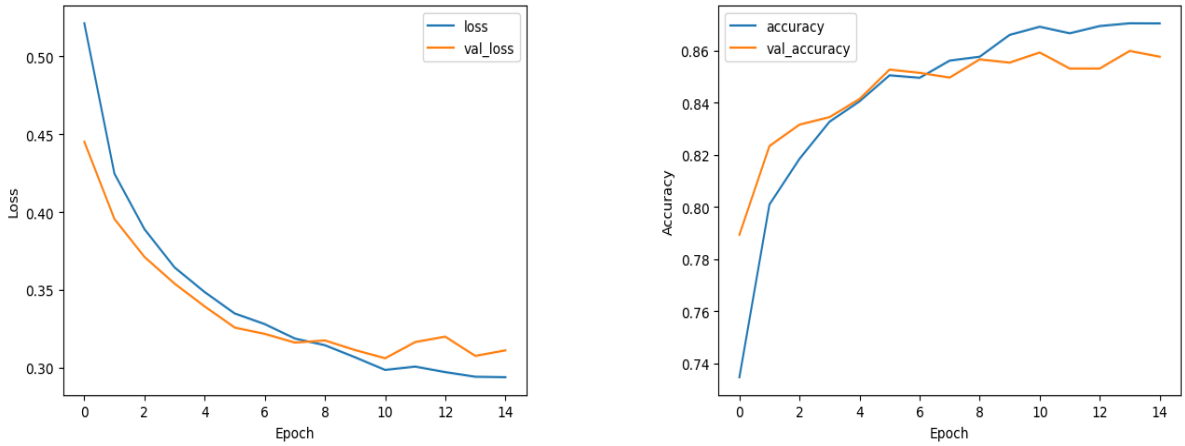
Ultimately, after conducting a myriad of experiments and meticulously analyzing the results,

we identified the optimal model configuration—a finely-tuned ensemble of hyperparameters, optimizer settings, and model architecture modifications. This configuration, which involved leveraging the EfficientNetB0 architecture with five layers removed, an Adam optimizer with a learning rate of 0.001, a dropout rate of 0.8, and a batch size of 128, emerged as the pinnacle of our experimentation efforts, delivering superior performance on our mushroom classification task.

Through meticulous experimentation and empirical validation, we have laid a solid foundation for future research and advancements in mushroom classification and machine learning alike.

0.6 Results

In our study, we conducted extensive experiments to optimize the performance of our mushroom classification model by varying different hyperparameters. Initially, we utilized a learning rate of 0.0001 with the Adam optimizer, which demonstrated promising progress in learning. Both validation and training losses exhibited favorable trends, with the validation loss decreasing while the training loss remained below 90%, indicating effective learning without significant overfitting. Despite these promising signs, the model's performance did not reach optimal levels.



(a) Training loss vs validation loss

(b) Training accuracy vs validation accuracy

Figure 4: Experiment 1 Metrics: Comparison of loss and accuracy

Subsequently, we pursued further refinement by decreasing the learning rate to 0.00002, allowing the model more flexibility to explore smaller steps in the parameter space. Experimenting with different dropout rates and batch sizes, we identified a configuration with a dropout rate of 0.7 and a batch size of 64, which yielded the best results within our hyperparameter search. However, this configuration still resulted in misclassifying 303 poisonous mushrooms as edible and 309 edible mushrooms as poisonous.

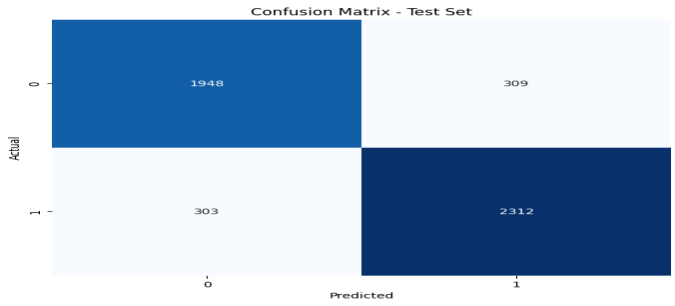
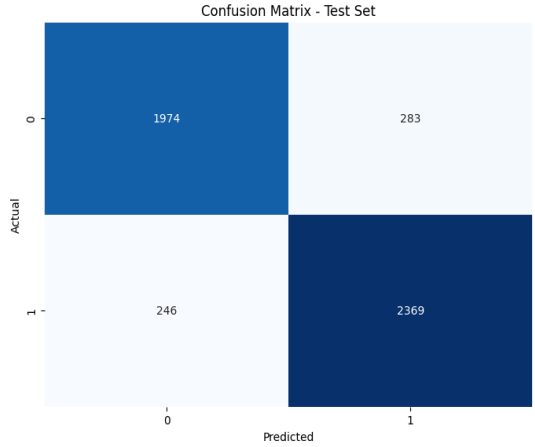


Figure 5: Experiment 2 Metrics: Confusion matrix

To fine-tune the model further, we adopted an Adam optimizer with a learning rate of 0.001

and increased the dropout rate to 0.8 while maintaining a batch size of 64. This adjustment led to a substantial improvement, reducing misclassifications to 283 edible and 246 poisonous mushrooms. This configuration emerged as the best model in terms of overall accuracy.



(a) Confusion matrix

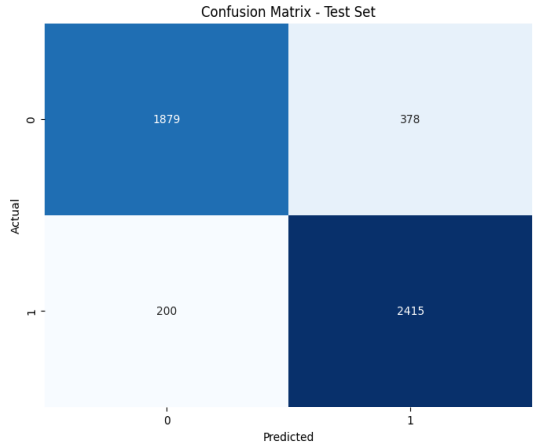
Classification Report

Class	precision	recall	f1-score	support
Class 0	0.89	0.87	0.88	2257.0
Class 1	0.89	0.91	0.9	2615.0
Class accuracy	0.89	0.89	0.89	0.89
Class macro avg	0.89	0.89	0.89	4872.0
Class weighted avg	0.89	0.89	0.89	4872.0

(b) Classification report

Figure 6: Experiment 3 Metrics: Best model for accuracy

However, our primary concern was to develop a model capable of correctly classifying poisonous mushrooms, prioritizing the avoidance of misclassifying them as edible. Further exploration led us to a final model configuration with a dropout rate of 0.8, Adam optimizer with a learning rate of 0.001, and a reduced batch size of 48. This model exhibited remarkable performance, misclassifying only 200 poisonous mushrooms as edible, the lowest among all tested models.



(a) Confusion matrix

Classification Report

Class	precision	recall	f1-score	support
Class 0	0.9	0.83	0.87	2257.0
Class 1	0.86	0.92	0.89	2615.0
Class accuracy	0.88	0.88	0.88	0.88
Class macro avg	0.88	0.88	0.88	4872.0
Class weighted avg	0.88	0.88	0.88	4872.0

(b) Classification report

Figure 7: Experiment 4 Metrics: Best model to find poisonous

The area under the Receiver Operating Characteristic (ROC) curve, often abbreviated as AUC-ROC, is a measure of the model's ability to discriminate between positive and negative

classes across all possible classification thresholds. In our case, an AUC-ROC value of 0.96 indicates that our mushroom classification model performs exceptionally well in distinguishing between poisonous and edible mushrooms.

To elaborate, the ROC curve plots the true positive rate (sensitivity) against the false positive rate (1 - specificity) for different threshold values. As the threshold varies, the true positive rate and false positive rate change accordingly, resulting in a curve that represents the trade-off between sensitivity and specificity.

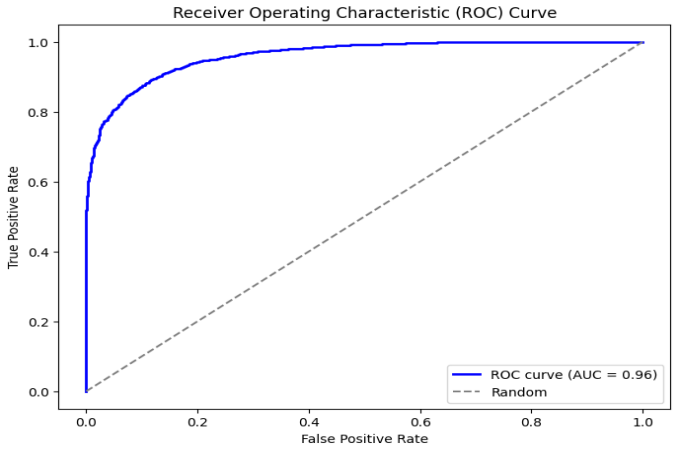


Figure 8: Experiment 4 Metrics: ROC curve

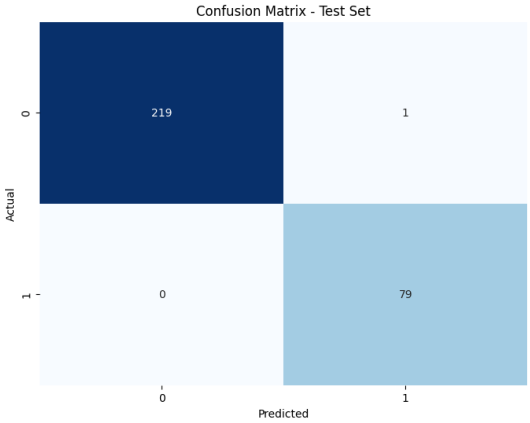
Moreover, our research contextualized the challenges posed by our dataset, which encompassed a wide range of mushroom species with diverse colors, shapes, and characteristics. Unlike previous studies that often relied on augmented datasets with limited variability, our dataset presented a more formidable task, yet our model, leveraging state-of-the-art EfficientNetB0 architecture and transfer learning, achieved impressive classification accuracy.

Our study underscores the potential of transfer learning and advanced neural network architectures in classifying mushrooms into edible and poisonous categories. Furthermore, it highlights the broader applicability of such models in the classification of other toxic vegetation and fruits, potentially serving as a valuable tool for survival guides during outdoor activities like trekking, hiking, or exploration.

0.6.1 Comparing our model with previous model with very high accuracy

In addition to evaluating our mushroom classification model on our dataset, we also assessed its performance on another dataset containing only three species of edible mushrooms and two species of poisonous mushrooms. A previous study reported an accuracy of 98.5% and a recall of 98.79% on this dataset, achieved by dividing it into a 90:10 ratio for training and testing. However, this approach, while yielding high accuracy and recall, may lead to model overfitting due to the presence of similar-looking and augmented images in both training and testing sets.

To address this concern, we adjusted the data division by allocating 70% for training, 15% for validation accuracy checking, and 15% for the test set. Remarkably, our model achieved

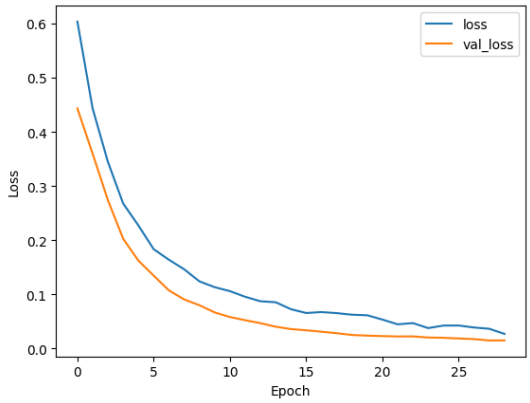


(a) Confusion matrix for Thailand dataset

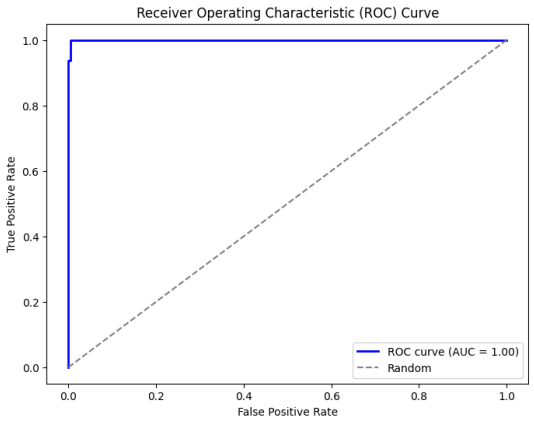
Classification Report

	precision	recall	f1-score	support
Class 0	1.0	1.0	1.0	220.0
Class 1	0.99	1.0	0.99	79.0
accuracy	1.0	1.0	1.0	1.0
macro avg	0.99	1.0	1.0	299.0
weighted avg	1.0	1.0	1.0	299.0

(b) Classification report for Thailand dataset



(c) Loss curve for Thailand dataset



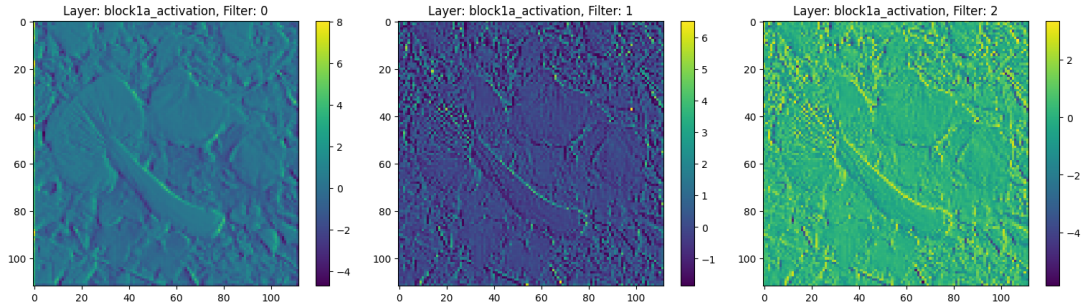
(d) ROC curve for Thailand dataset

Figure 9: Plots for Thailand dataset

an accuracy of 99.60% with a recall of 100%, successfully classifying all poisonous images as poisonous. In contrast, the previous study misclassified two poisonous images as edible, highlighting the potential consequences of using models trained on imbalanced or inadequately validated datasets in real-life scenarios.

0.7 Discussion

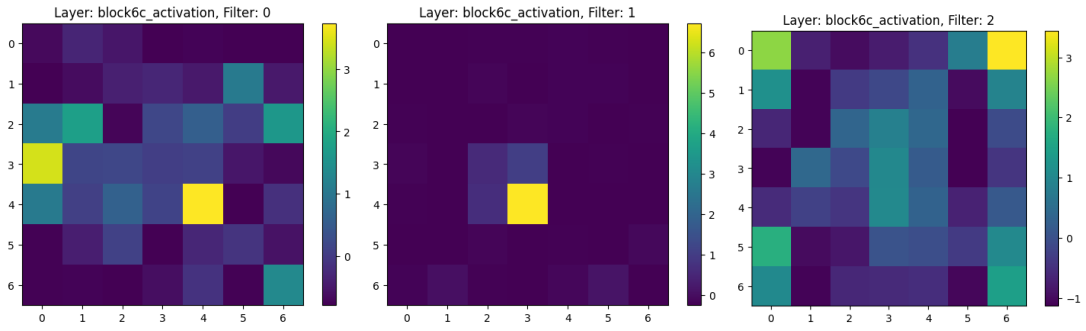
The results of our experimentation demonstrate the efficacy of transfer learning and hyperparameter optimization in developing a robust mushroom classification model. By systematically exploring different configurations, we were able to improve the model's performance and mitigate misclassifications, particularly focusing on correctly identifying poisonous mushrooms. Even though we were able to get a good result with such challanging dataset,deep learning faces numerous challenges, that includes overfitting, vanishing/exploding gradients, lack of interpretability, and data efficiency[**?**]. We tried to see what the filters are seeing in convolution layer. We took 3 filters from beginning layer, middle layer and last layer. The filters from first convolution layers were looking to find global pattern in the images. It tries to learn from the whole picture.



(a) Filter 1 (b) Filter 2 (c) Filter 3

Figure 10: Filters from the first convolutional layer

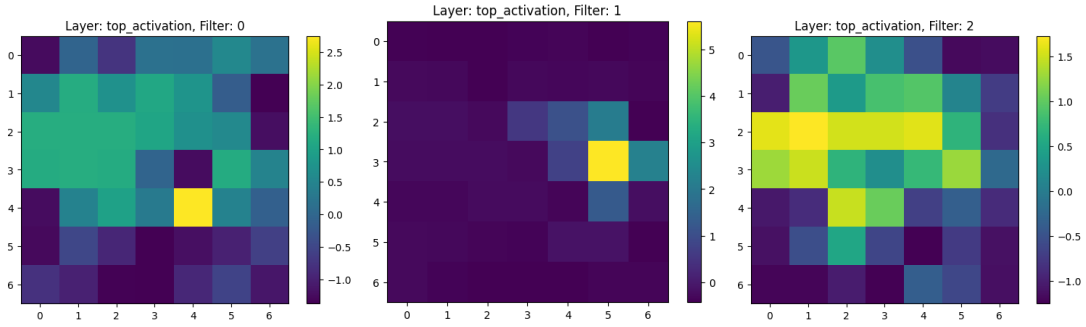
Similarly, 3 filters from block 6 convolution layer were viewed those patterns are hard to be understood by humans. This shows that the deep learning has the challange of interpretability.



(a) Filter 1 (b) Filter 2 (c) Filter 3

Figure 11: Filters from the sixth convolutional layer

We also tried to visualise what the top activation layer is seeing before deciding to classify the mushroom. Here as well interpretation is difficult.



(a) Filter 1 (b) Filter 2 (c) Filter 3

Figure 12: Filters from the top activation layer

The comparison of our model's performance on the additional dataset underscores the importance of appropriate data division and validation techniques in assessing model generalization and robustness. While the previous study reported impressive accuracy and recall rates, our adjustment to the data division revealed potential vulnerabilities in the model's classification capabilities. By allocating a portion of the dataset for validation accuracy checking and adopting a more balanced data division approach, we were able to achieve superior performance, accurately classifying all poisonous mushrooms and mitigating the risk of misclassification. This emphasizes the critical role of rigorous validation and evaluation methodologies in ensuring the reliability and efficacy of machine learning models in real-world applications.

The utilization of EfficientNetB0, a state-of-the-art convolutional neural network architecture, played a pivotal role in achieving high classification accuracy. Its ability to capture intricate features and patterns in mushroom images, despite their diverse colors and shapes, underscores its suitability for complex classification tasks.

Our study also sheds light on the challenges posed by diverse and unaltered datasets, contrasting with previous studies that often relied on augmented datasets for training. Despite these challenges, our model demonstrated competitive performance, showcasing the robustness of transfer learning in handling real-world datasets with inherent variability.

Furthermore, our research highlights the potential practical applications of such models beyond mushroom classification. The development of a mobile application incorporating this technology could serve as a valuable tool for outdoor enthusiasts, providing guidance on identifying edible and poisonous vegetation and fruits during outdoor expeditions or in survival situations.

Overall, our study contributes to the growing body of research on machine learning applications in mycology and underscores the potential for broader applications in related fields such as food safety, environmental conservation, and public health. Future research could explore additional techniques for dataset augmentation, further refine model architectures, and extend the classification framework to encompass a broader range of toxic vegetation and fruits for enhanced utility in real-world scenarios.

0.8 Conclusion

In conclusion, our study demonstrates the efficacy of transfer learning with the EfficientNetB0 architecture in optimizing mushroom classification. By fine-tuning the pre-trained model on a diverse dataset of mushroom images, we achieved superior classification accuracy and recall, accurately distinguishing between edible and poisonous mushrooms. Additionally, our model's performance on an independent dataset highlights its robustness and generalization capabilities, outperforming previous studies and mitigating the risk of misclassification. These findings underscore the importance of leveraging advanced architectures and fine-tuning techniques in machine learning model development, paving the way for enhanced applications in food safety, environmental conservation, and public health initiatives. Overall, our research contributes to advancing machine learning methodologies in mycology and underscores the potential of automated classification systems in addressing real-world challenges.

0.9 Future-Works

While our current study has made significant strides in optimizing mushroom classification through transfer learning and the EfficientNetB0 architecture, there remain several avenues for future research and improvement. The following are some key areas that warrant further exploration and development:

Expansion of Dataset Size and Class Balance: Increasing the number of images in each species class can significantly enhance the robustness of the classification model. It's essential to ensure not only a balanced number of images between the two main classes (edible and poisonous) but also within each species class. Addressing the imbalance in the number of images per species can help mitigate the potential bias towards more frequently represented species and improve the model's ability to classify less represented species accurately.

Integration of Generative Adversarial Networks (GANs): Implementing GANs to generate synthetic mushroom images presents a promising approach to augmenting the original dataset. By synthesizing additional images for underrepresented species, researchers can alleviate the imbalance in the dataset and provide the model with more examples to learn from, thus improving its classification performance.

Combination of Original and GAN-Generated Datasets: Combining datasets generated from GANs with the original dataset offers an opportunity to enhance model performance and generalization. By leveraging the diverse characteristics captured in both datasets, future studies can train more robust classification models capable of accurately distinguishing between different mushroom species. While this overlaps with the integration of GANs, it emphasizes the importance of incorporating diverse data sources to address dataset imbalances.

Utilization of Higher Resolution Images: Incorporating higher resolution images into the dataset can potentially enhance the classification results by providing more detailed information for the model to learn from. Future research efforts could focus on acquiring and preprocessing higher resolution mushroom images to improve the model's ability to discern subtle features and textures.

By addressing these areas for future work, researchers can further advance the field of mushroom classification and contribute to the development of more accurate, robust, and generalized classification models with broader applications in food safety, environmental conservation, and public health initiatives.

```
github link: https://github.com/Harithelamin/Is-poisonous

The code used for training the model:
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
from tensorflow.keras import layers, models, optimizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from efficientnet.tfkeras import EfficientNetB0
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import roc_curve, auc



np.random.seed(42)
tf.random.set_seed(42)

# Define paths to the training, validation, and test directories
train_dir = r'E:\Amazon\ml\original splitted\train'
test_dir = r'E:\Amazon\ml\original splitted\test'
val_dir = r'E:\Amazon\ml\original splitted\val'


# Define parameters for data generators
batch_size = 128
target_size = (224, 224)  # Target size for input images

# Create ImageDataGenerator instances with data augmentation and
   normalization for training and validation
train_datagen = ImageDataGenerator(
    rescale=1./255,  # Normalize pixel values to [0,1]
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

val_datagen = ImageDataGenerator(rescale=1./255)  # Only rescale
   for validation data

```

```
# Create train_generator and val_generator
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=target_size,
    batch_size=batch_size,
    class_mode='binary',  # Assuming binary classification
    shuffle=True
)

val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=target_size,
    batch_size=batch_size,
    class_mode='binary',
    shuffle=False
)

# Create ImageDataGenerator instance for test data
test_datagen = ImageDataGenerator(rescale=1./255)  # Only rescale for test data

# Create test_generator
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=target_size,
    batch_size=batch_size,
    class_mode='binary',
    shuffle=False
)

# Load pre-trained EfficientNetB0 model without top layers
base_model = EfficientNetB0(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Freeze the layers of the pre-trained model except the last few convolutional layers
for layer in base_model.layers[:-5]:  # Adjust the number of layers to freeze as per your requirement
    layer.trainable = False

# Add custom dense layers
x = base_model.output
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dense(256, activation='relu')(x)
x = layers.Dropout(0.8)(x)
predictions = layers.Dense(1, activation='sigmoid')(x)  # Binary
```

```
    classification, so using sigmoid activation
87
88 # Combine base model with custom top layers
89 model = models.Model(inputs=base_model.input, outputs=predictions)
90
91 # Specify the learning rate
92 learning_rate = 0.001
93
94 # Compile the model with custom learning rate
95 optimizer = Adam(learning_rate=learning_rate)
96 model.compile(optimizer=optimizer, loss='binary_crossentropy',
    metrics=['accuracy'])
97
98 # Compile the model
99 model.compile(optimizer=optimizer, loss='binary_crossentropy',
    metrics=['accuracy'])
100 # Define early stopping callback
101 early_stopping = EarlyStopping(monitor='val_accuracy', patience=10)
102
103 # Train the model with early stopping
104 history = model.fit(
105     train_generator,
106     steps_per_epoch=len(train_generator),
107     epochs=70,
108     validation_data=val_generator,
109     validation_steps=len(val_generator),
110     callbacks=[early_stopping]
111 )
112
113 # Plot training history
114 plt.plot(history.history['accuracy'], label='accuracy')
115 plt.plot(history.history['val_accuracy'], label='val_accuracy')
116 plt.xlabel('Epoch')
117 plt.ylabel('Accuracy')
118 plt.legend()
119 plt.show()
120
121 # Plot training history for loss
122 plt.plot(history.history['loss'], label='loss')
123 plt.plot(history.history['val_loss'], label='val_loss')
124 plt.xlabel('Epoch')
125 plt.ylabel('Loss')
126 plt.legend()
127 plt.show()
128
129 # Predict on test set
```

```
print("Predicting on test set...")
test_pred_probs = model.predict(test_generator)
test_preds = (test_pred_probs > 0.5).astype(int)


# Confusion matrix for test set
cm_test = confusion_matrix(test_generator.classes, test_preds)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_test, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Test Set')
plt.show()

# Evaluate the model on validation set
print("Evaluating the model on validation set...")
val_loss, val_accuracy = model.evaluate(val_generator)
print(f"Validation Loss: {val_loss:.4f}, Validation Accuracy:
    {val_accuracy:.4f}")


# Generate classification report
class_report = classification_report(test_generator.classes,
    test_preds, target_names=['Class 0', 'Class 1'],
    output_dict=True)

# Convert classification report to DataFrame
report_df = pd.DataFrame(class_report).transpose()

# Visualize classification report as a table with adjusted font
    size and 2 decimal places
plt.figure(figsize=(10, 6))  # Adjust figure size as needed
plt.axis('off')  # Turn off axis
table = plt.table(cellText=report_df.round(2).values,  # Round to 2
    decimal places
                  rowLabels=report_df.index,
                  colLabels=report_df.columns,
                  cellLoc='center',
                  loc='center',
                  colWidths=[0.2]*len(report_df.columns),
                  bbox=[0, 0, 1, 1])

# Adjust font size
table.auto_set_font_size(False)
table.set_fontsize(10)  # Adjust font size as needed

```

```
plt.title('Classification Report')

plt.tight_layout()
plt.show()

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(test_generator.classes,
    test_pred_probs)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC =
    %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='gray', linestyle='--',
    label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()

# Choose an image from the validation set
img = val_generator[0][0][0]  # Assuming batch size is 128 and you
    want to visualize the first image
true_label = val_generator[0][1][0]  # Assuming batch size is 128
    and you want to get the true label of the first image

# Make a prediction on the image
prediction = model.predict(np.expand_dims(img, axis=0))
predicted_class_index = np.argmax(prediction)
predicted_class_label = class_labels[predicted_class_index]  #
    Assuming class_labels is defined

# Define function to compute activation map for a specific layer
    and filter
def compute_activation_map(model, img, layer_name, filter_index):
    # Get the specified layer by name
    layer = model.get_layer(layer_name)
    # Create a submodel that outputs the activation of the
    specified layer
    submodel = tf.keras.models.Model(inputs=model.inputs,
    outputs=layer.output)
    # Compute the activations for the input image
    activations = submodel.predict(np.expand_dims(img, axis=0))
    # Extract the activations for the specified filter
```

```
    activations_filter = activations[:, :, :, filter_index]
    # Normalize the activations
    activations_filter -= activations_filter.mean()
    activations_filter /= activations_filter.std() + 1e-5
    return activations_filter[0]

# Define function to plot activations for a specific layer and filter
def plot_activations(activation_map, layer_name, filter_index):
    plt.imshow(activation_map, cmap='viridis')
    plt.title(f'Layer: {layer_name}, Filter: {filter_index}')
    plt.colorbar()
    plt.show()

# Define filters to visualize: 3 filters from first layer, 3 from 40th, 3 from last layer
filters_to_visualize = [('block1a_activation', i) for i in range(3)] + \
                       [('block6c_activation', i) for i in range(3)] + \
                       [('top_activation', i) for i in range(3)]

# Iterate over filters and visualize activations
for layer_name, filter_index in filters_to_visualize:
    # Compute the activation map for the chosen layer and filter
    activation_map = compute_activation_map(model, img, layer_name, filter_index)
    # Plot the activation map
    plot_activations(activation_map, layer_name, filter_index)

# Display the true class label and the predicted class label
print("True class label:", true_label)
print("Predicted class label:", predicted_class_label)
```

Code 1: Python code for training the model