Deep Learning Final Project Technical Report

Tsolisou Dafni, Filippidou Thalassini, Psallidas Kyriakos

[1]Department of Computer Science and Telecommunications, National and Kapodistrian University of Athens

Abstract

Medical image classification is crucial for improving diagnosis and treatment, especially when human analysis is time-consuming and less precise compared to Computer-Aided Diagnosis (CAD) systems. Our study focused on classifying microscopic fungi images, addressing a significant yet frequently overlooked health threat posed by fungal infections. We evaluated the effectiveness of transfer learning using four pre-trained models (VGG16, ResNet50, EfficientNetb0, and ViT16) using various metrics like balanced accuracy, MCC, F1-score, and confusion matrices. We also compared deep learning models with traditional machine learning algorithms such as Logistic Regression, Naive Bayes, and Random Forest. To ensure accurate model assessment, we implemented cross-validation for deep learning models to better assess their performance on unseen data. Notably, our findings reveal that EfficientNet outperformed all other models, achieving a remarkable balanced accuracy of 0.9 when augmented. Additionally, we employed GRAD-CAM for model explainability and visualized the Attention mechanism for VIT. These findings underscore the significant potential of deep learning models in medical image classification and their crucial role in addressing critical healthcare challenges.

Introduction

Fungi represent a vast and diverse group of eukaryotic organisms that exist as unicellular yeasts or filamentous molds. [1] Historically, humans have harnessed the unique properties of fungi for food production, in cases such as bread, wine and fermented dairy products. In contemporary contexts, the industrial applications of fungi have expanded significantly, notably in biotechnology and pharmaceuticals. However, while some fungi offer beneficial properties, several species can instigate a spectrum of diseases, from superficial and cutaneous infections to more severe systemic conditions [1].

According to epidemiology reports [2] fungal infections are responsible for over 1.5 million deaths globally per year. Immunocompromised patients can often be affected more severely, leading to high mortality rates among them. [3] During the COVID-19 pandemic there was a surge in the number of opportunistic fungal infections. [4] Despite all these, there is a worldwide tendency to overlook fungal infections from public health considerations, which leads to less clinical awareness and a lack of standardized guidelines for their diagnosis and treatment [5].

Traditional methods of diagnosis such as microscopy and culture, often require prolonged periods of time for accurate results. The classification of a fungal infection needs to be made in a laboratory by a specialized biologist known as a mycologist. This complexity, combined with the fact that symptoms caused by fungal infections are ambiguous, contributes for patients to be diagnosed late, thus leading to an inconsistency in their outcomes. [6] The role of early and accurate diagnosis in the aggressive containment of the fungal infection at the initial stages is crucial since it can prevent the development of a life-threatening situation [7] .

Given that diagnosis often requires a specialized mycologist to examine microscopic images of fungi, the utilization of Computer Aided Diagnosis (CAD) techniques can substantially accelerate this process. Particularly, the application of Deep Learning (DL), especially Computer Vision methods, has the potential to increase accuracy and reduce classification time of an infection. Trained on mycologic images, a DL algorithm can detect subtle patterns and anomalies often imperceptible to the human eye, potentially allowing for earlier intervention.

This study seeks to evaluate the efficacy of prominent computer vision algorithms using a moderately-sized academic dataset of mycological images. While the dataset's original publication [8] extensively leveraged deep learning methods for classification, our goal is to provide a comparative analysis of various models' performance on this specific collection.

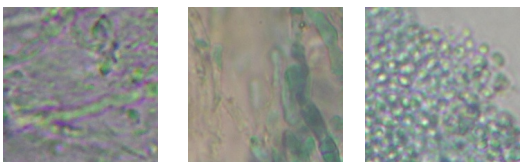
The power of deep learning algorithms lies on the availability of vast amounts of labelled data, which is not always easy to acquire in the medical domain. Transfer learning is a powerful tool which can be used to harness the wealth of knowledge residing in related but distinct source domains, thereby alleviating the heavy reliance on vast quantities of target-domain data. This approach offers a practical solution to the challenge of data scarcity. Instead

of starting from scratch, transfer learning leverages the knowledge and feature representations learned by the pre-trained model from its original task. By fine-tuning or adapting the pre-trained model on the new data, it can swiftly adapt its knowledge to the nuances of the target problem. Moreover, it's worth noting that we already have numerous excellent image classification models that we can use and adapt to address new and diverse problems effectively. These existing models serve as powerful starting points, leveraging the collective knowledge and expertise of the machine learning community, making it easier to achieve remarkable results in various image-related tasks.

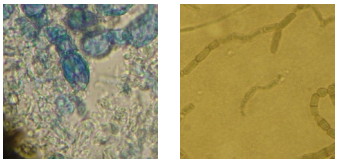
Methodologies

Data-set Description & Exploratory Data Analysis

This project's dataset is obtained from a curated selection extracted from an initial pool of 3,000 raw and unlabeled RGB mages showcasing various fungal infections induced by yeasts, molds. These images were provided by the Laboratory of Electron Microscopy and Microanalysis (LEMM) at Victoria University, Australia. Subsequently, to enhance data quality, a pre-processing methodology, which involves segmenting the images into smaller patches and eliminating patches containing artifacts was applied to the dataset as described in the paper from which the dataset originates [8]. The final data set is available and titled: "Microscopic Fungi Image - DeFungi Dataset" at Kaggle. It consists of a total of 6,801 images and five classes as showcased in the class example images of Figure 11(a). Furthermore, the dataset is split into predefined "train", "test" and "validation" image directories.



(a) Candida albicans (b) Aspergillus niger (c) Trichophyton rubrum



(d) Trichophyton mentagrophytes (e) Epidermophyton floccosum

Figure 1: *Example microscopy image per class*

After loading the data, we examined the class distribution within each of the splits, as illustrated in Figure 10. It is clear from these observations that the distribution of samples in both the test and validation splits of the data set is imbalanced. Consequently, while the model was trained on a balanced data-set, its evaluation does not account for this imbalance. We considered two solutions to address this issue. The first solution involves using weighted evaluation metrics that take into account the class distribution in the test and validation sets while keeping the data set unchanged. The second solution is to merge the entire data-set and utilize stratified splits along with a weighted loss function to incorporate cost-sensitive learning. The second approach was deemed more suitable since it "respects" the current known distribution of class labels in the whole data set, thus we merge the data set and end up with class distributions of Candida albicans = 28%, Aspergillus niger = 22%, Trichophyton rubrum = 17% Trichophyton mentagrophytes = 17% and finally Epidermophyton floccosum = 17%. Following the assignment of class labels derived from their respective class directories, we proceeded to visualize ten example instances from each class and their corresponding 3-channel histograms, along with their mean pixel intensity value as well as image size, as showcased in Figures 12, 13, 14, 15 and 16. As observed from these example images, the data set contains images of varying average intensity and two distinct sizes: 224 by 224 pixels and 179 by 179, with the latter making up only 8% of the total data set (Figure 11(b)). To apply Principal Component Analysis (PCA), we initially flattened the 3-D image arrays (size 224x224x3) into 1-D vectors, resulting in vectors containing 150,528 elements for each image. Furthermore, we created a second flattened data set by first applying a Grayscale transformation to the original RGB images, resulting in 1-D vectors with 50,176 elements for each image. Following that we applied PCA with $n = 700$ & $n = 950$ components on the gray and RGB data-sets respectively. Finally, we restricted the number of resulting PCA components in the flattened data sets to the number that corresponds to 95% of the total variance and applied Uniform Manifold Approximation and Projection (UMAP) to visualize the dependencies present within the aforementioned feature space, the results of which are showcased in the Results section [9].

Classical Machine Learning

We believe it is important to establish a baseline for machine learning performance in order to gain a clearer understanding of the advantages of employing deep learning architectures for overcoming the

complex task of multi-class image classification.

To conduct our experiments, we utilized the gray-scale data set with all images resized to 224 by 224 pixels utilizing bilinear interpolation from the *PIL* library [10]. We chose the gray-scale version over RGB data set due to its significantly reduced feature count. This choice was made to overcome the potential impact of very large feature vectors on the run time and performance of classical machine learning algorithms.

Prepossessing of the data set for classical ML Consisted of a stratified train, validation, and test split methodology using the *scikit-learn* library [11]. Specifically, we allocated 70% of the data for training, 15% for validation, and another 15% for testing purposes, maintaining the class balance of the original dataset. To enhance the shape patterns of the images overshadowed due to different contrast levels we utilized histogram equalization from the *openCV*[12] library on the training, validation, and test splits, which improves contrast by stretching the intensity values of the image across the entire dynamic range as showcased for an example image in (Figure 2). Feature extraction is an extremely important step in

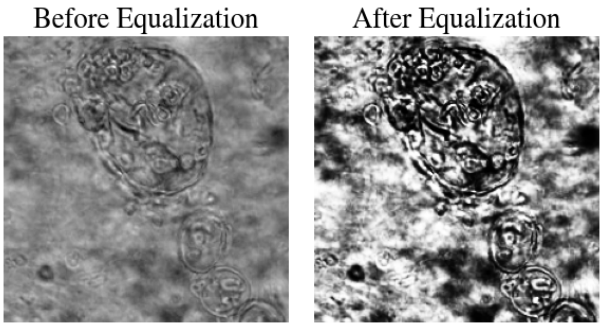


Figure 2: *Histogram Equalization*

machine learning. We opted to extract and utilize as features the distribution of gradient orientations in each image with the Histogram of Oriented Gradients method (HoG) from the *PIL* library [cite PIL]. This allows us to capture changes in pixel intensities and subsequently local shape and edge information as presented in (Figure 3). After extracting the HoG features, we standardized each dataset by fitting a standard scaler to the training set initially. Consequently, the dimensions of the final training, validation, and test sets are as follows: (4760, 54756), (510, 54756), and (510, 54756), respectively.

Classifiers utilized for this multi-class problem are all part of the sci-kit libray. Specifically, they are the Naive Bayes (NB) classifier to establish a baseline. Additionally, we explored the performance of multi-class Logistic Regression (LR) and Random Forest (RF) classifiers. For the LR classifier, we introduced

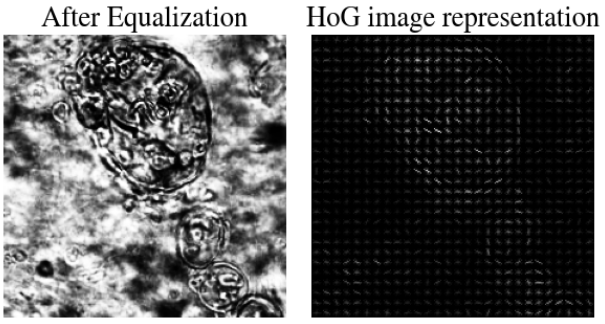


Figure 3: *Histogram of Oriented Gradients*

regularization through the parameter C. To fine-tune these classifiers, we leveraged the *optuna* library [13], utilizing Bayesian optimization. For the RF classifier we tuned key hyperparameters, including the number of estimators, tree depth, and minimum samples per split and leaf. The optimization process aimed to maximize the Matthews correlation coefficient (MCC) on the validation set. The MCC metric was chosen as our primary selection criterion due to its ability to yield high scores when the classifier accurately predicts all class instances. It is defined as follows:

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

Deep Learning

During the recent years, developments in the area of deep learning models have dramatically increased the type and number of problems that could be solved by neural networks. Although deep learning techniques are not new, it has exponentially become more useful as the amount of available data has increased and computer infrastructure both in terms of hardware and software, has improved. Notably, the intersection of deeply layered neural networks and the use of GPUs has significantly accelerate their execution time [14].

An area which has greatly been impacted by this developed is that of computer vision, particularly the task of image classification. In contrast to traditional machine learning algorithms which require various handcrafted features from the image, deep learning algorithms are designed to operate directly on images. Both feature extraction and classification are done automatically by the model which learnsto detect the images having similar objects. This alleviates the need for the researchers to manually search for the best features in order to optimize learning [15, 14].

A general deep learning architecture contains a combination of some neural layers like input layers, convolution, fully connected layers, sequence layers,

activation layers, normalization, dropout, pooling, output layers and many more. Only the input and output layers' values are easily accessible by us, the rest are called hidden layers. It is in these hidden layers where the majority of important work happens and because of them complex data may be modelled.

In this study two specialized kind of neural networks are test, three convolutional (ConvNets) and one Vision Transformer. Each type of model is based on different kind of layers and architectural patterns in order to tackle the image classification task. The steps we followed to train and test our models on mycological images are described in detail in this section.

Preprocessing of the data set for Deep Learning To handle the preprocessing and dataset management tasks, we utilized two essential built in PyTorch classes: *Datasets* and *Dataloaders*[16]. In particular, we employed the *ImageFolder* method on the directory containing the total of 6,801 RGB images, each organized within sub-directories corresponding to their respective classes. This approach allowed us to generate an iterable dataset that contains both the image data and their corresponding labels. Furthermore, this dataset allows for the definiton of a prepossessing transformation applied to the images. We opted to utilize the preprocessing transformation standard for most pre-trained models, that is:

1. Resize the images to 224 by 224 using bilinear interpolation
2. Transform the image to tensor
3. Adjust the channels by subtracting 0.485, 0.456, and 0.406 from each channel, then divide the result by 0.229, 0.224, and 0.225 for each respective channel to standardize them.

Following that we initiated Dataloaders to handle our two main training approaches: a stratified cross-validation training to establish an average estimate of the models performance across different datasets and finally training on a normal training, validation split to create the final model.

To conduct cross-validation, we utilized the scikit-learn library to create an 85% training and 15% testing data split. Following this, we implemented a stratified 5-fold split on the training dataset, which provided us with the requisite indices for segmenting the training data-set into five distinct subsets. We then constructed separate data loaders for each split, with a batch size of 32 for each data loader. To prevent model bias during back-propagation, we incorporated data shuffling into the training data loaders, ensuring that the images were presented in a randomized order.

For the final model training, we generated a secondary split comprising training and validation sets, allocated at 85% and 15%, respectively, from the initial 85% training dataset. These datasets were then loaded into two distinct data loaders, utilizing the same configuration parameters as those utilized during the cross-validation phase.

It's important to note that we also employed an alternative preprocessing transformation for the training dataset, which introduced data augmentations to introduce variability in the dataset and prevent over-fitting on the training set. This approach was utilized for a distinct final model training session and its steps are:

1. Resize the images to 224 by 224 using bilinear interpolation
2. Flip the image horizontally (50% chance)
3. Rotate the image randomly to up to 10 degrees left or right
4. Increase or decrease the brightness and contrast by up to 20% and the saturation and hue by up to 10%.
5. Transform the image to tensor
6. Adjust the channels by subtracting 0.485, 0.456, and 0.406 from each channel, then divide the result by 0.229, 0.224, and 0.225 for each respective channel to standardize them.

The custom Pytorch training loop we utilized is illustrated in a schematic representation in (Figure 4). First, we move our model to the GPU device (P100), then perform a forward pass utilizing iterative batches of 32 images from the training data set This includes predicting labels for each batch, calculating the training loss utilizing the cross entropy loss function:

$$L(y, p) = -\sum_{i=1}^{N} y_i \cdot \log(p_i)$$

, where y_i is the true probability distribution over classes (a one-hot encoded vector indicating the true class). And p_i is the predicted probability distribution over classes after applying a softmax function applied to the model's logits output. It's crucial to emphasize that the logits are individually weighted by 0.7258, 0.9285, 1.1696, 1.1726, and 1.1942 respectively. This weighting strategy is employed to incorporate class balance-sensitive learning. Next, we initiate backpropagation from the model's known output layer value, using the chain rule to compute gradients for the weights of all layers. We then utilize the Adam optimizer with a learning rate of 0.0001, which is progressively reduced by a factor of 0.01 in each step. This helps us update the model's weights iteratively, aiming to minimize the loss function. Finally, the current model's weights are frozen and the

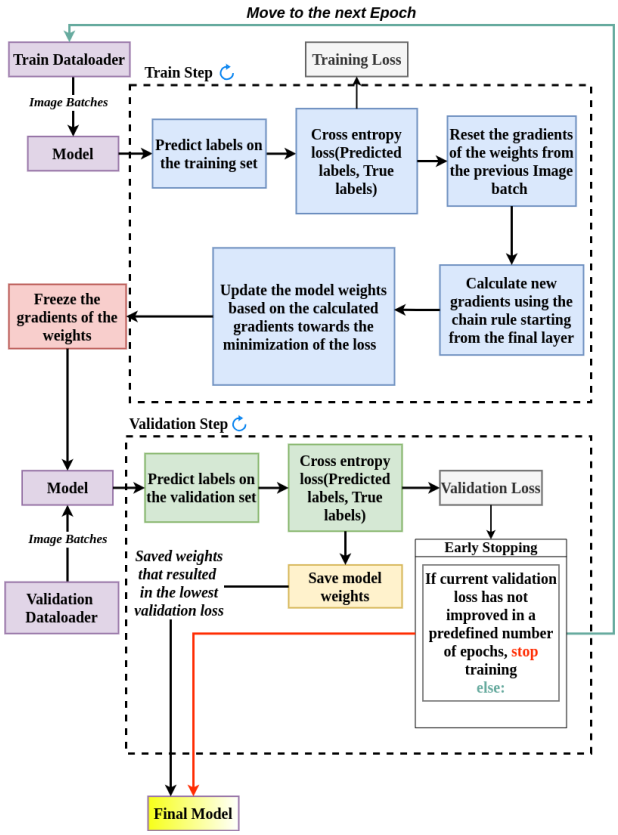


Figure 4: *Schematic representation of our custom Pytorch training loop with early stopping callback*

validation Dataloader provides image batches for the model to make predictions on the validation set, from these predictions a validation loss is calculated in the same manner as the training step. Subsequently, we save the current model weights in memory and assess whether the validation loss has shown improvement compared to the previous epoch. If the validation loss remains unchanged or increases for five consecutive epochs, we terminate the training process. Finally, once the training is complete, we proceed to make predictions on the test set using the same methodology employed for the validation set. This allows us to calculate on the test set, the classification metrics: F1 score, Balanced Accuracy, and Matthews Correlation Coefficient (MCC).

When it comes to cross-validation, we employ the mentioned training methodology independently for each of the five stratified splits, utilizing the Dataloaders as described in the preprocessing section. However, after each split, the model is reset by loading its default weights from training on the ImageNet dataset. This reset is performed to ensure that the model begins training from the same initial weights for each validation fold.

Transfer Learning As mentioned the usage of pre-trained models on very large datasets like ImageNet (which contains 1.2 million images with 1000 categories) for initialization or as fixed feature extractors for a task of interest has become very popular in image classification. [15] Nowadays, rarely do people train from scratch large models since there are many pre-trained models freely available which produce superior accuracy in a large variety of tasks.

Transfer learning can be done in two ways:

- **Finetuning the ConvNet** Instead of randomly initializing the weights of a model, the network is initialized with pre-trained weights and then trained for a few epochs on the dataset at hand with the usual training loop.
- **ConvNet as fixed feature extractor** The weights of a model are frozen for all the network except the final fully connected layer which are replaced with random numbers. During training only these final weights are updated to fit the dataset at hand.

For our implementation, we made use of four pre-trained models taken directly from Pytorch, to which we loaded the default pre-trained weights. The architectures of the model was kept the same except for the output layers which were reset to 5 neurons, as many as the number of classes in our dataset. We chose to fine-tune each model to our training set because the fixed feature extractor performed poorly in our case. The steps we followed to load and use the pre-trained models are the following:

1. Load the appropriate model using the torchvision module and the default pre-trained weights.
2. Reset the final layer according to our number of classes.
3. Obtain the transforms done during the initial training of the model and use them for transforming our data-set.

The four models we tested are briefly described below:

VGG16 A convolutional neural network architecture introduced by the Visual Geometry Group (VGG) from the University of Oxford [17] in 2014. It was developed in an effort to investigate how depth influences performance and trained on the ILSVRC-2012 dataset (ImageNet). It's results confirmed the notion that depth in visual representations is of great importance.

The vgg16 version is comprised by 16 weight layers which includes 13 convolutional and 3 fully connected (Figure 5). The architecture also includes a series of convolutional filters with a very small receptive field (3x3), which made it possible to increase

the depth of the model. These architectural characteristics make up for 138 million trainable parameters. While the model employs max-pooling layers to down-sample feature maps, these are not applied after every convolutional layer. This design ensures superior pattern recognition while maintaining computational efficiency.

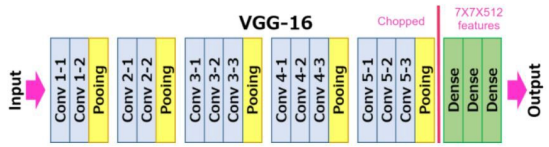


Figure 5: *The architecture of the VGG16 model.*

ResNet , or Residual Networks, emerged in 2015 as a significant advancement in neural network architecture primarily to tackle the vanishing gradient problem. This challenge arises when training very deep networks, as gradients diminish as they propagate backward through numerous layers, hindering learning. To overcome this, ResNet introduced skip connections, or residual connections, which allowed gradients to flow directly between layers. This innovation effectively mitigated the vanishing gradient issue, facilitating the training of exceptionally deep networks. The ability to learn intricate features across multiple layers propelled ResNet into a foundational architecture in deep learning, underpinning numerous breakthroughs in the field [**kaming2015resnet**]. In our study we utilized the ResNet50 architecture which comprises of 5 convolution layers along with the average pooling and final classifier layer.

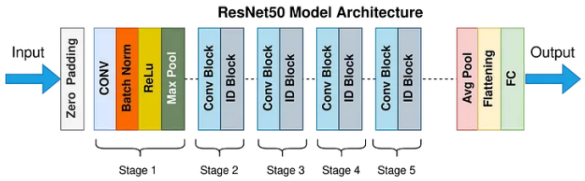


Figure 6: *The architecture of the Resnet50 model.*

EfficientNet , introduced in 2019, is a novel neural network architecture that addresses the challenges of scaling ConvNets for enhanced accuracy. It's creators highlighted the importance of balancing network dimensions, including width, depth, and resolution. They proposed a pragmatic solution, the compound scaling method, which uniformly adjusts these dimensions using a fixed scaling coefficient. For example, to increase computational resources by a factor of 2^N, depth is scaled by αN, width by βN, and image size by γN, where α, β, γ are determined through a grid search on the smaller model. They constructed a baseline network EfficientNetb0 by utilizing a multi-objective neural architecture search, which optimizes both accuracy and FLOPS (Floating Point Operations Per Second) and scaled it up using the coumpound scaling method to create a family of networks. EfficientNets achieved state-of-the-art accuracy on ImageNet and also showed good perfomance in transfer learning tasks while substantially reducing parameters and FLOPS, making it a significant advancement in deep learning [18]. In our study we utilized EfficientNetb0 which achieved a 77.1% Top-1-Accuracy in the ImageNet dataset with only 5.3 million parameters.

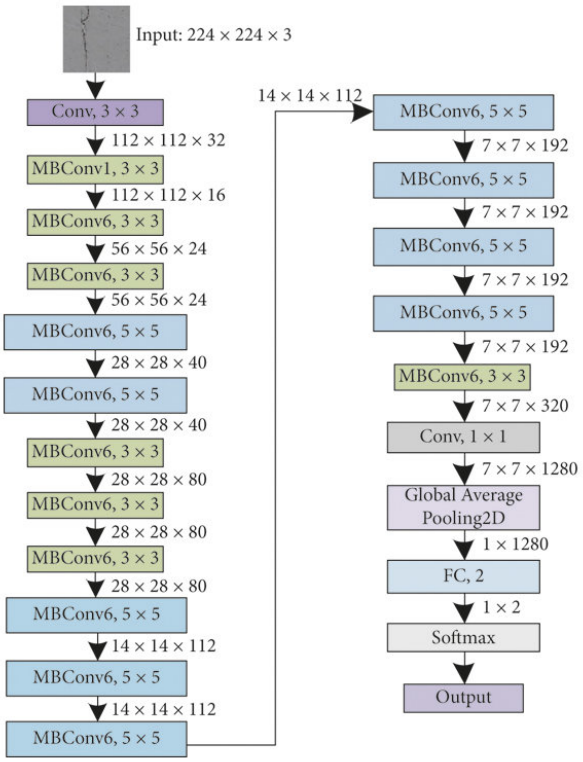


Figure 7: *The architecture of the EfficientNetb0 model.*

Vision Transformer Base 16 (ViT) In 2021, the groundbreaking paper titled "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale" introduced the concept of vision transformers [19]. The primary aim was to apply the state-of-the-art transformer architecture, which had demonstrated exceptional performance in NLP tasks, to the tasks of computer vision. The novel components of vision transformers relate to how they handle input images until the creation of positional embeddings. Following this stage, the architecture closely resembles that of NLP. Initially, the input image is divided into image patches, after which a convolutional neural network (CNN) with a kernel size equal to the patch size extracts features from each patch, resulting in embeddings. Subsequently, self-attention is com-

puted among all image patches, allowing for global context awareness. A high level schematic of the ViT architecture sourced from the original paper is presented in (Figure 8).

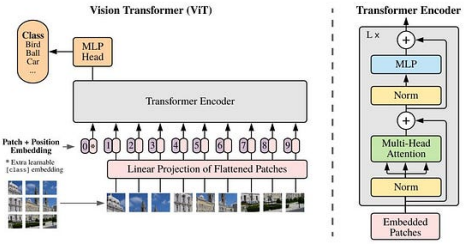


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

Figure 8: *[source: [19]]*

When comparing vision transformers to CNNs architectures for image tasks, such as VGG16, Resent and Efficient Net, The primary conceptual distinction lies in the fact that CNNs employ moving kernels of fixed size to aggregate local information, with the receptive field typically expanding as we delve deeper into the network. In contrast, vision transformers capture global relationships from the early stages of the network. From this distinction, several key points emerge. Vision transformers, due to their immediate global focus, often demand larger datasets and longer training times compared to CNNs, which adopt a hierarchical approach, gradually transitioning from local to global focus. However, when provided with large data resources, vision transformers excel in learning intricate patterns and establishing complex relationships, thus they are well-suited for integration into a transfer learning framework.

Model Explainability with GradCam & Attention

As deep learning models become more complex the computations they perform and the decisions they reach are getting harder for humans to interpret. For this reason these models are often referred to as black boxes. Making their decision process more transparent and interpretable is an important concept, which has been recently discussed more and more. This is what the concept of model explainability aims to do, shed light into the decision making process of an intelligent system.

The benefits from this practice include easier identification of failure modes, build trust between AI systems and users and finally for humans to learn from AI in cases when it is significantly stronger (e.g. the game of GO). [20] Explainability is particularly important in the field of healthcare, where a wrong decision can have adverse effects on the lives of patients and healthcare providers.

GradCam A popular method for model explainability is Gradient-weighted Class Activation Mapping (Grad-CAM)[20], which was introduced in 2017 at the International Conference on Computer Vision (ICCV). It is a valuable tool particularly for ConvNets used in image processing tasks which acts as a class-discriminative localization technique that generates visual explanations (heatmaps) for any CNN-based network without requiring architectural changes or re-training.

Grad-CAM uses the gradient information flowing into the last convolutional layer of the CNN to assign importance values to each neuron for a particular decision of interest, thus explaining the output layer decisions. Through the visualizations it produces in input images we can inspect the parts of the image which influenced a particular decision. Since deeper layers in the network usually capture high-level features, selecting such a layer can give insights into which high-level features are most important for a particular prediction.

A visual depiction of the way it works can be seen in Figure 9. Given an image and a class of interest as input, a forward propagation through the CNN part of the model is performed. Then through task-specific computations a raw score for the category is obtained. The gradients are set to zero for all classes except the desired class, which is set to 1. This signal is then backpropagated to the rectified convolutional feature maps of interest, which are combined to compute a heatmap which represents the areas where the model focuses to make the particular decision. Finally, the heatmap is multiplied pointwise with the guided backpropagation to get Guided Grad-CAM visualizations which are both high-resolution and concept-specific.

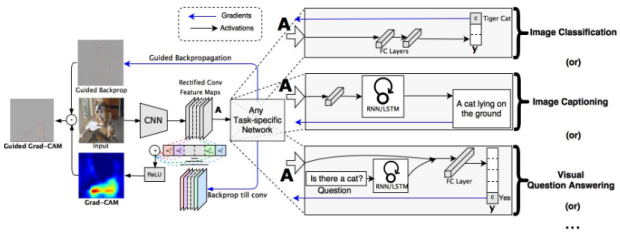


Figure 9: *An overview of how Grad-CAM works [source [20]].*

In our implementations we applied Grad-Cam to our CNN-based models using the Pytorch gradcam library which is freely available on github [21]. Only the models trained with data augmentation were used for explainability. The library's usage is fairly easy, we just need to provide the Grad-Cam constructor with a model, a target layer, the target class we

want to inspect and an input image preprocessed appropriately. Using these inputs it can produce a heatmap and use it as an overlay on the original image.

We also chose to apply smoothing (aug-smooth) to reduce noise in the class activation results. Aug-smooth increases the run-time by a factor of 6 and applies random augmentations to the input image to better center the activation results around the object.

Attention To gain a deeper understanding of the functioning of the Vision Transformer Base 16 model, we employed a visualization approach to inspect the intermediate outputs during the model's forward pass. This examination was conducted using the trained model on an augmented data-set that yielded optimal results. To achieve this analysis, we utilized PyTorch Hooks which allows us to capture information of the intermediate layers in the model architecture without modifying the forward pass function.

Specifically, we registered a forward pass hook on the single convolutional layer to capture the output image patches derived from the original input image. Furthermore, we employed twelve forward pass hooks, each associated with one of the twelve multi-head attention layers. These hooks captured the attention outputs originating from the flattened embedded image patches.

Results & Discussion

PCA & UMAP

As depicted in Figure 19(b), the RGB dataset allows for a superior class separability among data points, with a cumulative explained variance of 54.5% in the first two Principal Components (PCs). In contrast, the grayscale dataset achieves a cumulative variance of 46.6% for its first two PCs. Analyzing the cumulative explained variance against the number of PCs reveals that the grayscale dataset reaches 95% variance with 680 components, while the RGB dataset accomplishes this with 869 components, as demonstrated in (Figure 19). Finally, the application of UMAP on the feature space spanned by the 680 and 869 components further verified that RGB images allow for greater class separability (Figure 19).

Performance of classical ML classifiers

Among the 54,756 HoG features that span the feature space, the baseline NB classifier, assuming independence, achieved the second-best performance. It resulted in scores of 0.27 for MCC, 0.39 for F1, 0.39 for Balanced Accuracy, and 0.41 for Accuracy (Figure 20(a)). The optimal Random Forest (RF) classifier outperformed all others, obtaining scores of 0.31 for MCC, 0.41 for F1, 0.42 for Balanced Accuracy, and 0.43 for Accuracy (Figure 20(c)). On the contrary, the optimal Logistic Regression (LR) classifier performed the least favorably, with scores of 0.19 for MCC, 0.35 for F1, 0.34 for Balanced Accuracy, and 0.35 for Accuracy (Figure 20(b)). Upon examining the confusion matrices illustrating the true versus predicted classes for all three classifiers (refer to Figures 20(d), 20(e), and 20(f)), we can verify that their prediction performance is sub-par. The above showcase the need of Deep Learning on such a complex task.

Cross Validation Scores

A comparison of the 5-fold average performance of the models on the same train-validation folds can be seen in Table 1 and Figure 21 which also includes the standard deviations. Overall, we observe that EfficientNetb0 demonstrates the best average perfomance while ResNet50 comes second followed by ViT. However, there is relatively little variation in the performance among the different models.

	VGG16	Eff.Net	ResNet50	ViT
n. Parameters	138.4M	5.3M	25.6M	86.6M
Balanced acc.	0.822	0.861	0.848	0.835
MCC	0.758	0.807	0.795	0.778
F1-score	0.803	0.847	0.835	0.822

Table 1: *Average Classification metrics of vision models on the stratified 5-fold validation splits*

Test Performance

The performance on the held-out test set of the final models which were trained on the dataset without data augmentation can be seen in Table 2. Table 3 compares the same performance of the models trained with data augmentation.

Strong performance across all four models can be observed. EfficientNet stands out with exceptional results, consistently achieving the highest scores in all three validation metrics for both the regular and augmented datasets. While Visual Transformers (ViT) demonstrated good performance in the regular dataset, it did not exhibit significant improvement when applied to the augmented dataset. Conversely, ResNet and VGG displayed substantial enhancements in all metrics when augmented data was incorporated.

It is clear from these two tables that data augmentation has a positive effect on learning and helps all models to generalize better on the test set. This is because with data augmentation each batch is al-

ways slightly different at each epoch, thus reducing overfitting on the training set.

We can also see this effect when comparing the training curves between these two cases. Figure 22 depicts the loss and balanced accuracy curves of the final models trained with and without data augmentation. It is readily apparent that in the second case the validation loss and balanced accuracy curves closely follow the corresponding ones of the train loss and balanced accuracy. Whereas in the first case they quickly diverge, leading the model to overfit faster.

Finally, comparing the test set confusion matrices (Figure 23) of the models when trained with data augmentation, we can better understand which classes are more likely to be misclassified. It is interesting that all models are performing extremely well for the H3, H5 and H6 classes, with EfficientNet again being the best one. On the other hand, all models seem to struggle more with H1 and H2 classes, which are often confused for one another.

This confusion on those two classes can possibly be attributed to the their characteristics and the examples found in each one. Observing Figures 12 and 13 we notice that some images are quite similar between these two classes, with no strong details to distinguish them. These examples indicate that their feature maps could resemble one another, thus leading the models to misclassify them.

	VGG16	Eff.Net	ResNet	ViT
n. Parameters	138.4M	5.3M	25.6M	86.6M
Balanced acc.	0.821	0.884	0.841	0.878
MCC	0.758	0.843	0.787	0.816
F1-score	0.803	0.871	0.824	0.852

Table 2: *Classification metrics of vision models on the test test*

	VGG16	Eff.Net	ResNet	ViT
n. Parameters	138.4M	5.3M	25.6M	86.6M
Balanced acc.	0.870	0.918	0.910	0.879
MCC	0.819	0.882	0.874	0.814
F1-score	0.854	0.905	0.901	0.850

Table 3: *Classification metrics of vision models with Data Augmentation on the test set*

Grad-Cam Heatmaps

In Figure 25 we can see the heatmaps Grad-Cam produced from the CNN-based models using one image per class from our dataset. To produce the heatmaps the class where each image should be classified was used as target. Based on these visualization we can see that in some cases the models do manage to discover the fungi and ignore the background, thus making its decision based on the shape of the microorganism. However, it is clear that in some cases, like class H1, the models were not able to figure out the shape of the fungi and classify it correctly.

It is also interesting that in cases where the fungi was discovered, like in the case of class H6, the models payed attention to approcimately the same parts of the image, but each one slightly different. Especially in the H5 class, where the fungi can clearly be distinguished, VGG16 and ResNet50 only care about one part of the microorganism's body, whereas EfficientNet looks at a larger part.

These findings can help us better understand the reasons why the models struggled more in some classes and visualize examples that confused them. By comparing them with examples that were correctly classified the decision process of models becomes more transparent to human's and can help researchers focus on ways to improve models.

ViT Attention Visualization results

We initiated our analysis by selecting an example image from the test set, at index 600, with dimensions of (224, 224, 3), as depicted in Figure 25(a). This image is then divided into 768 smaller image patches, each (14, 14) in size. These patches were subsequently processed through a single convolutional layer, responsible for extracting features from them as illustrated for the first 20 out of the 768 patches in Figure 25(b). It's worth noting that each of the 768 image patches is then transformed into an embedding vector, of size (1, 196 + 1). Notably, the first element in this vector represents the class token. The graphical representation of these flattened patch histograms can be observed in Figure 25(c).

Following this process, the resulting embedding vector is then subjected to a sequence of twelve consecutive layers employing multi-head attention mechanisms to capture dependencies between the 768 flattened Image patches, the (768,197) attention maps between all the elements of the patch embedding vectors for all patches and multi-head-attention layers are showcased in Figure 26.

Conclusions

Based on our findings we can safely say that Deep Learning models, especially when used in the context of transfer learning, can demonstrate outstanding performance in image classification tasks compared to traditional Machine Learning algorithms. Moreover, the DL models' generalization ability on unseen data from the same distribution is far superior, hav-

ing more than 40% better scores than the best ml classifier. This is in alignment with existing literature and findings from contemporary research.

Another important result of this study is the fact that EfficientNetb0 outperformed all other models, when trained both on the original and the augmented dataset, despite it having less parameters than the other models. This indicates that the multi-objective neural architecture search performed from its authors can be a promising method for the architectural design of neural networks.

These findings can be a starting point for further experimentation with different data augmentation strategies as well as model architectures. Hyperparameter tuning, which was not utilized in this study could also help in achieving better scores. Furthermore, taking advantage the results of the explainability methods can also boost overall performance.

References

[1] Michael R McGinnis and Stephen K Tyring. "Introduction to mycology". In: *Methods Gen. Mol. Microbiol* (1996), pp. 925–928.

[2] Felix Bongomin et al. "Global and multi-national prevalence of fungal diseases—estimate precision". In: *Journal of fungi* 3.4 (2017), p. 57.

[3] Emily Rayens and Karen A Norris. "Prevalence and Healthcare Burden of Fungal Infections in the United States, 2018". In: *Open Forum Infectious Diseases* 9.1 (Jan. 2022), ofab593.

[4] Rimesh Pal et al. "COVID-19-associated mucormycosis: an updated systematic review of literature". In: *Mycoses* 64.12 (2021), pp. 1452–1459.

[5] ML Rodrigues and JD Nosanchuk. *Fungal diseases as neglected pathogens: a wake-up call to public health officials. PLoS Neglect Trop D 14: e0007964*. 2020.

[6] Perry E Formanek and Daniel F Dilling. "Advances in the diagnosis and management of invasive fungal disease". In: *Chest* 156.5 (2019), pp. 834–842.

[7] Wenjie Fang et al. "Diagnosis of invasive fungal infections: challenges and recent developments". In: *Journal of Biomedical Science* 30.1 (2023), p. 42.

[8] Camilo Javier Pineda Sopo, Farshid Hajati, and Soheila Gheisari. "DeFungi: Direct mycological examination of microscopic fungi images". In: *arXiv preprint arXiv:2109.07322* (2021).

[9] Leland McInnes, John Healy, and James Melville. "Umap: Uniform manifold approximation and projection for dimension reduction". In: *arXiv preprint arXiv:1802.03426* (2018).

[10] Alex Clark et al. "Pillow (pil fork) documentation". In: *readthedocs* (2015).

[11] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *the Journal of machine Learning research* 12 (2011), pp. 2825–2830.

[12] Gary Bradski. "The openCV library." In: *Dr. Dobb's Journal: Software Tools for the Professional Programmer* 25.11 (2000), pp. 120–123.

[13] Takuya Akiba et al. "Optuna: A next-generation hyperparameter optimization framework". In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 2623–2631.

[14] S Suganyadevi, V Seethalakshmi, and K Balasamy. "A review on deep learning in medical image analysis". In: *International Journal of Multimedia Information Retrieval* 11.1 (2022), pp. 19–38.

[15] Manali Shaha and Meenakshi Pawar. "Transfer learning for image classification". In: *2018 second international conference on electronics, communication and aerospace technology (ICECA)*. IEEE. 2018, pp. 656–660.

[16] Adam Paszke et al. "Automatic differentiation in pytorch". In: (2017).

[17] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[18] Quoc V. Le Mingxing Tan. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *arXiv:1905.11946* (2019).

[19] Alexey Dosovitskiy et al. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *arXiv preprint arXiv:2010.11929* (2020).

[20] Ramprasaath R Selvaraju et al. "Grad-cam: Visual explanations from deep networks via gradient-based localization". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 618–626.

[21] Jacob Gildenblat and contributors. *PyTorch library for CAM methods*. `https://github.com/jacobgil/pytorch-grad-cam`. 2021.

Figures

For the sake of visual clarity, the remaining figures referenced in the methods & results sections of the report are showcased on the following pages: $11 - 27$.

Code Availability

The code to reproduce the results of the report and/or extend the analysis is available in this GitHub repository.

Data Availability

The data-set utilized in this study is available in this Kaggle link.

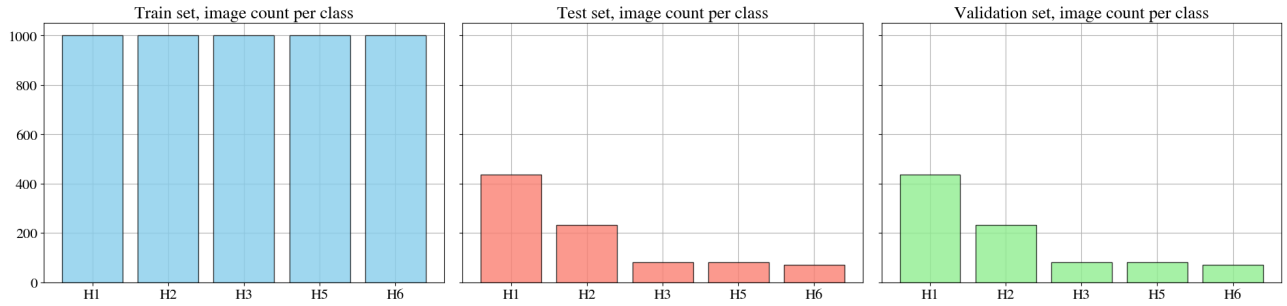
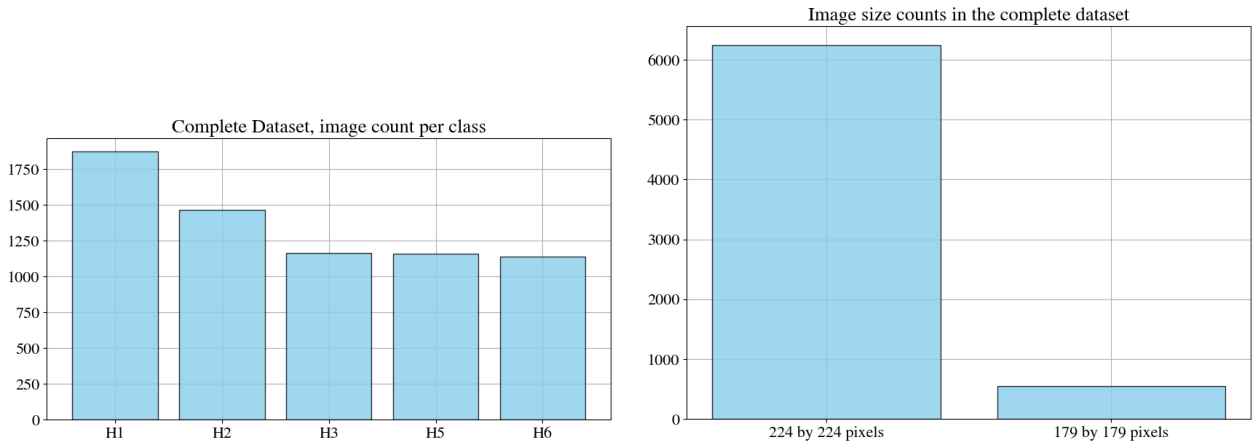


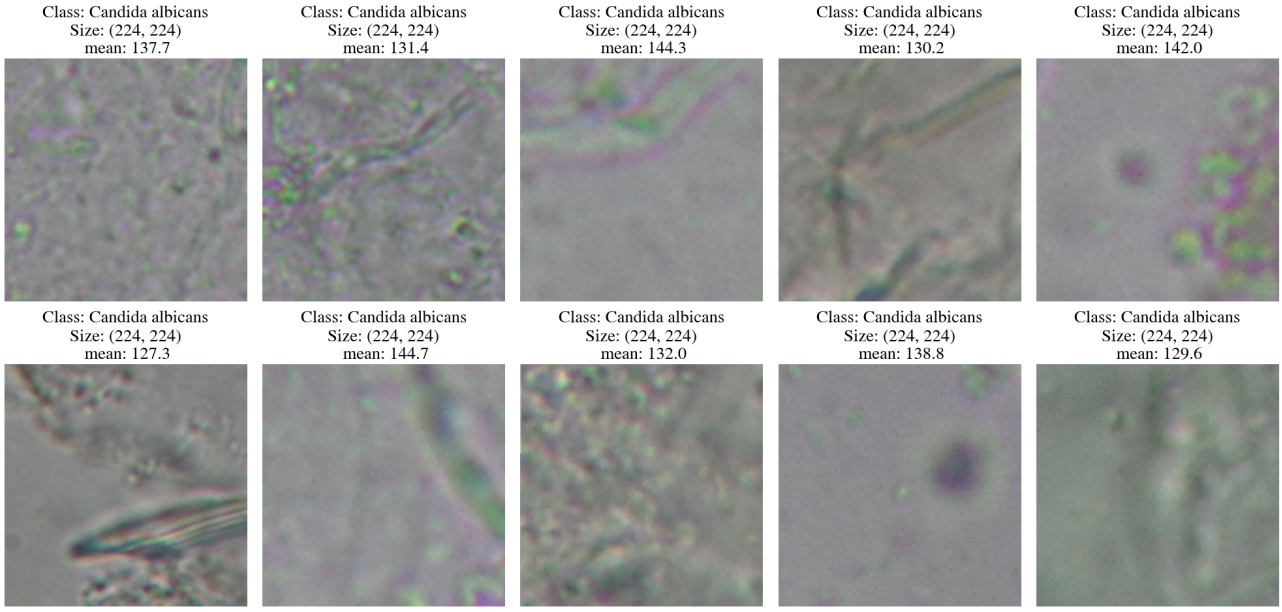
Figure 10: *Class counts balance per data-set split*



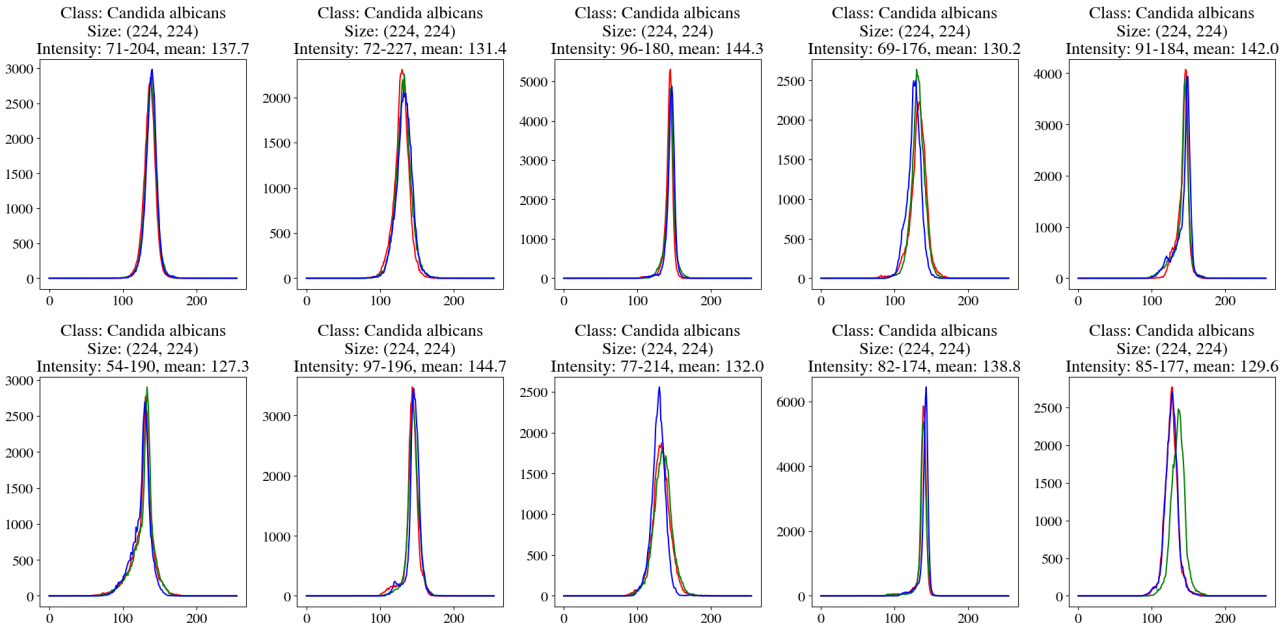
(a) Class counts

(b) Image size counts

Figure 11: *Total merged dataset class & Image size counts*

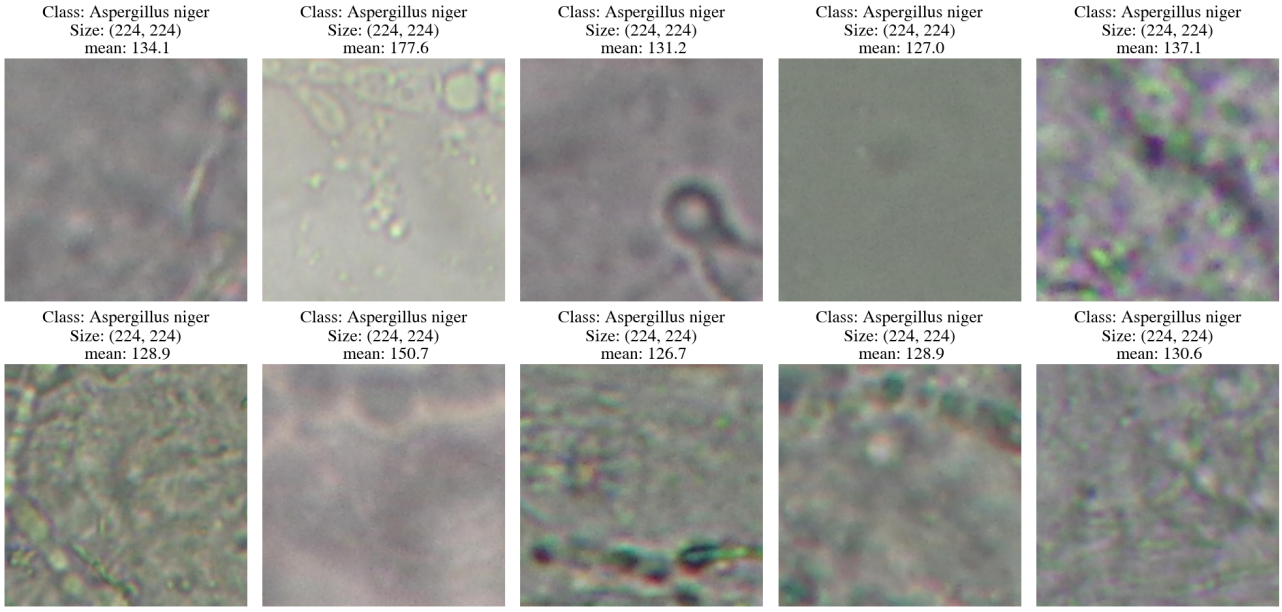


(a) Images

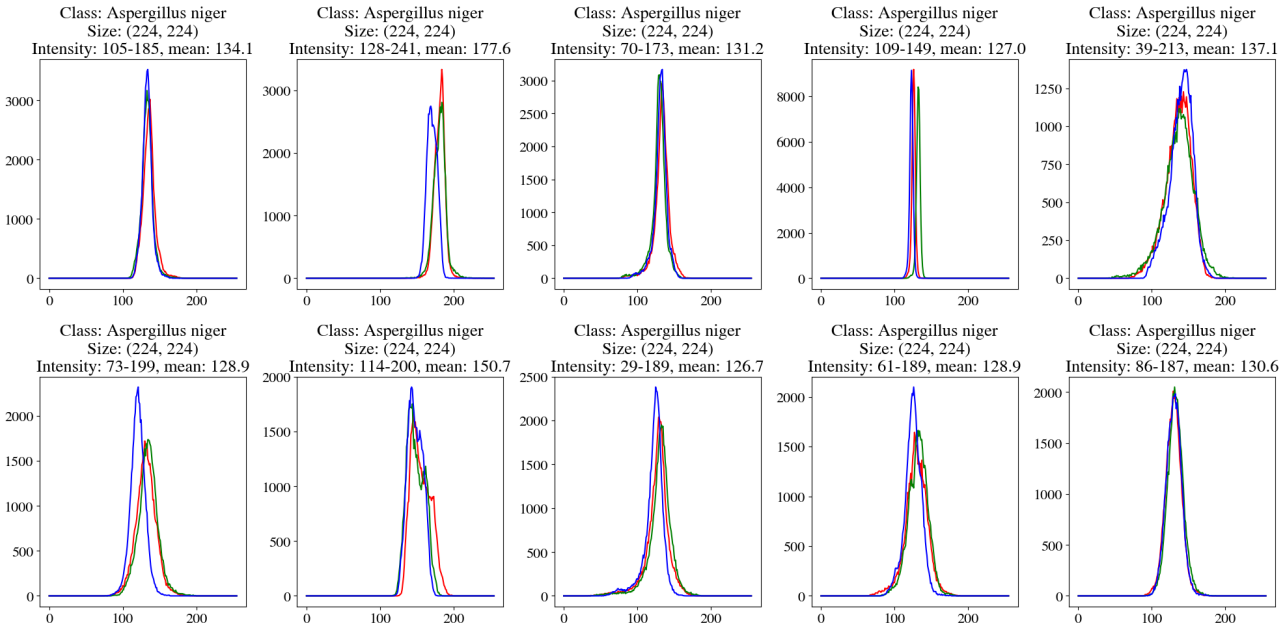


(b) Histograms

Figure 12: *Candida Albicans example images & RGB histograms*

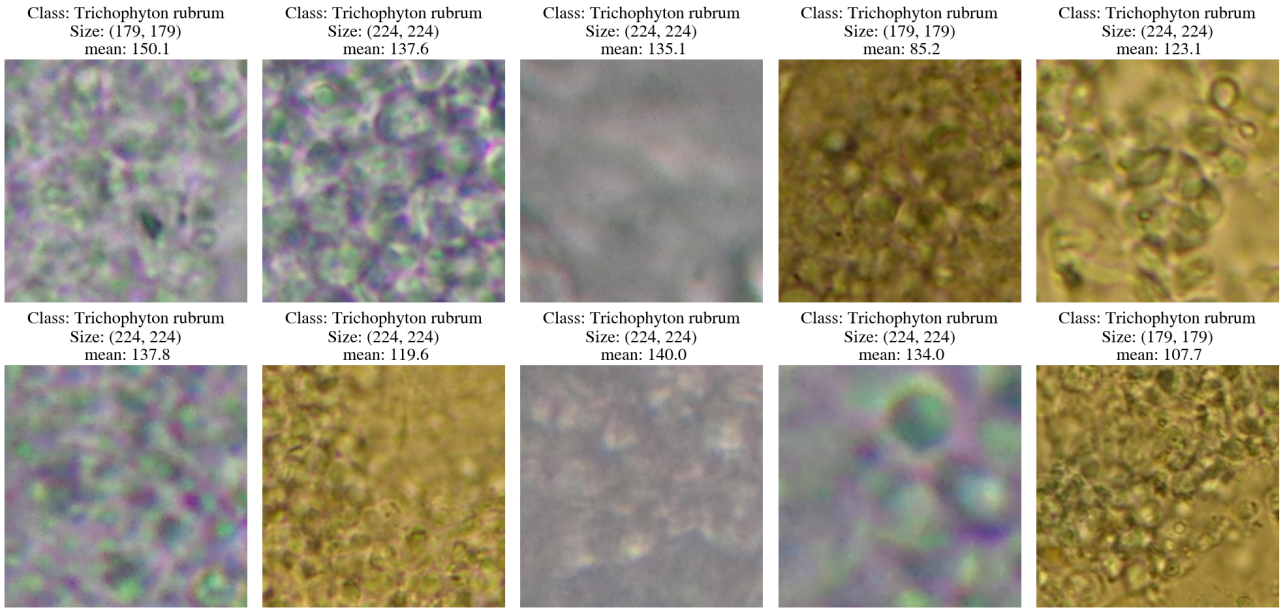


(a) Images

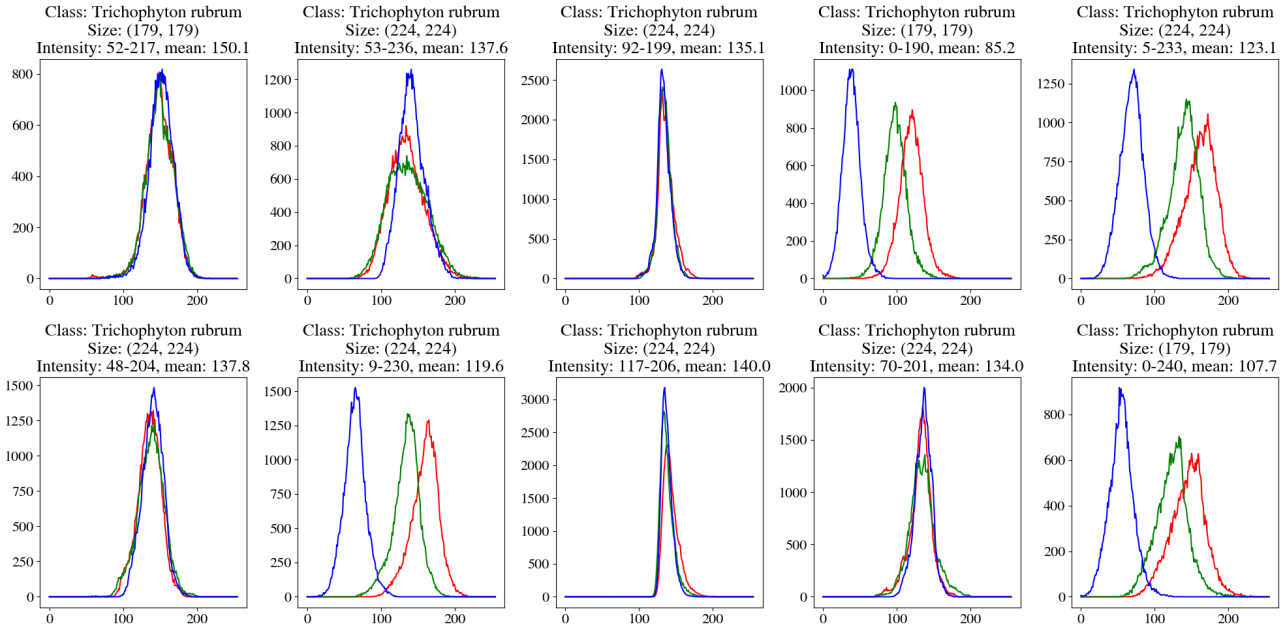


(b) Histograms

Figure 13: *Aspergillus Niger example images & RGB histograms*

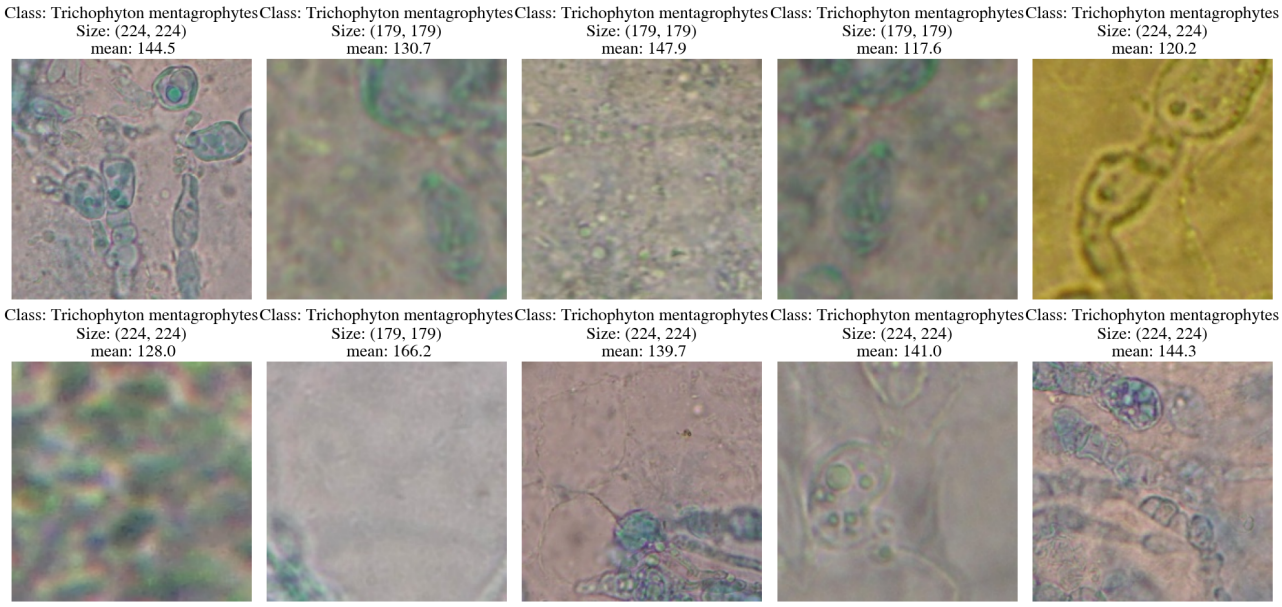


(a) Images

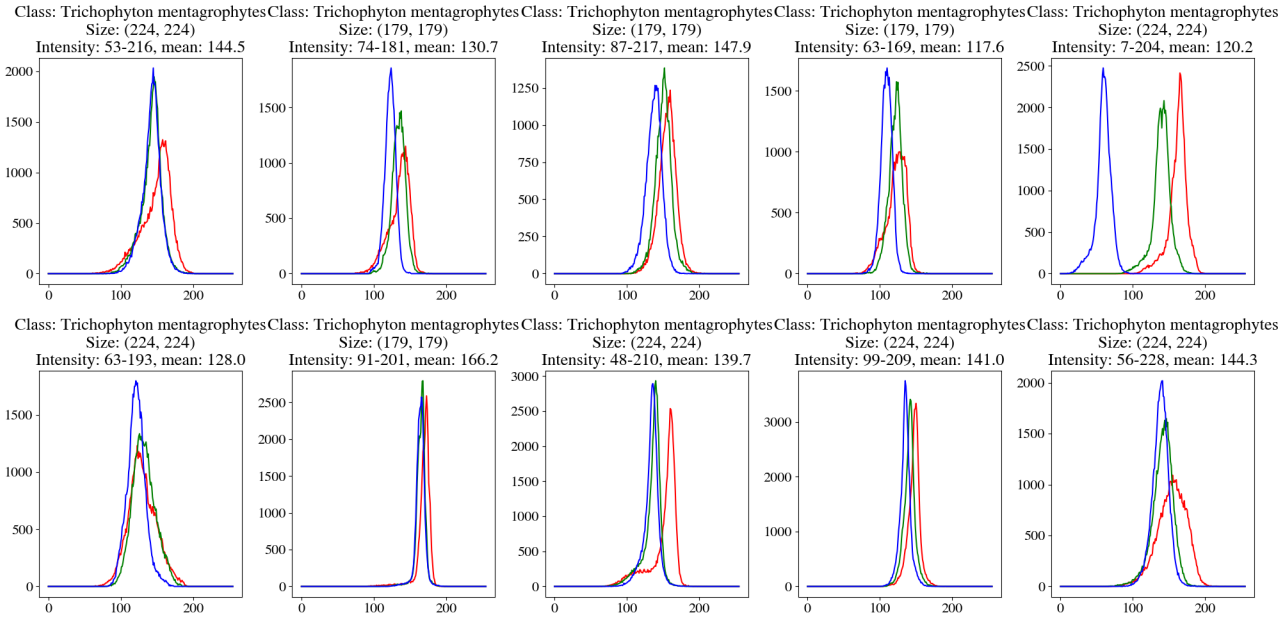


(b) Histograms

Figure 14: *Trichophyton rubrum example images & RGB histograms*

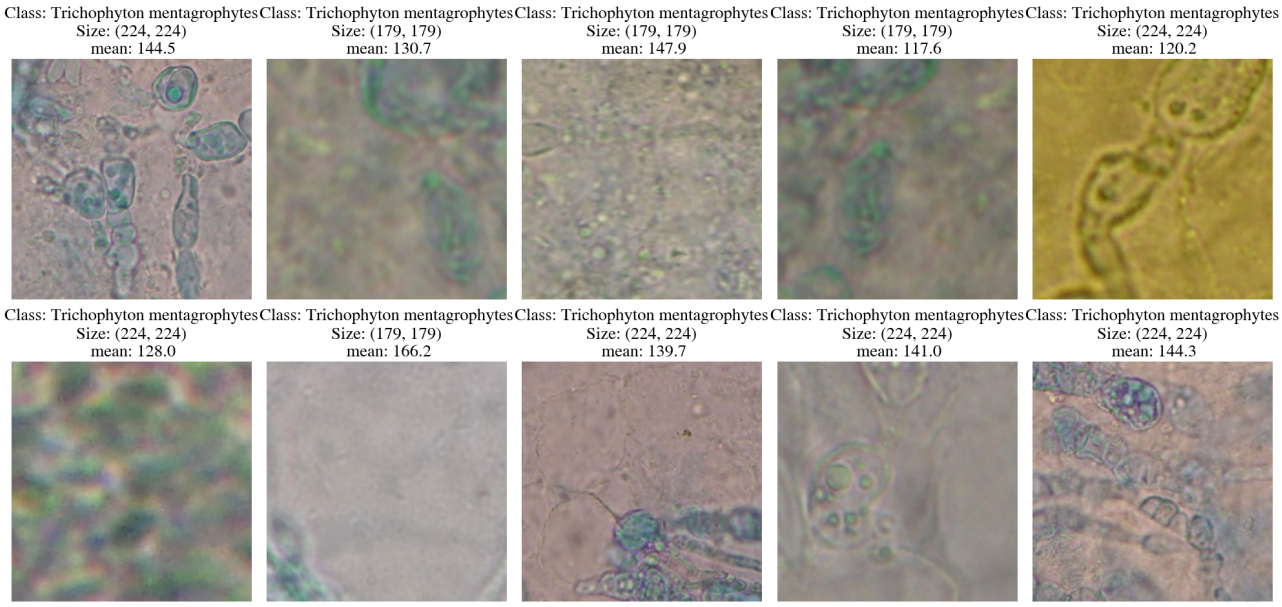


(a) Images

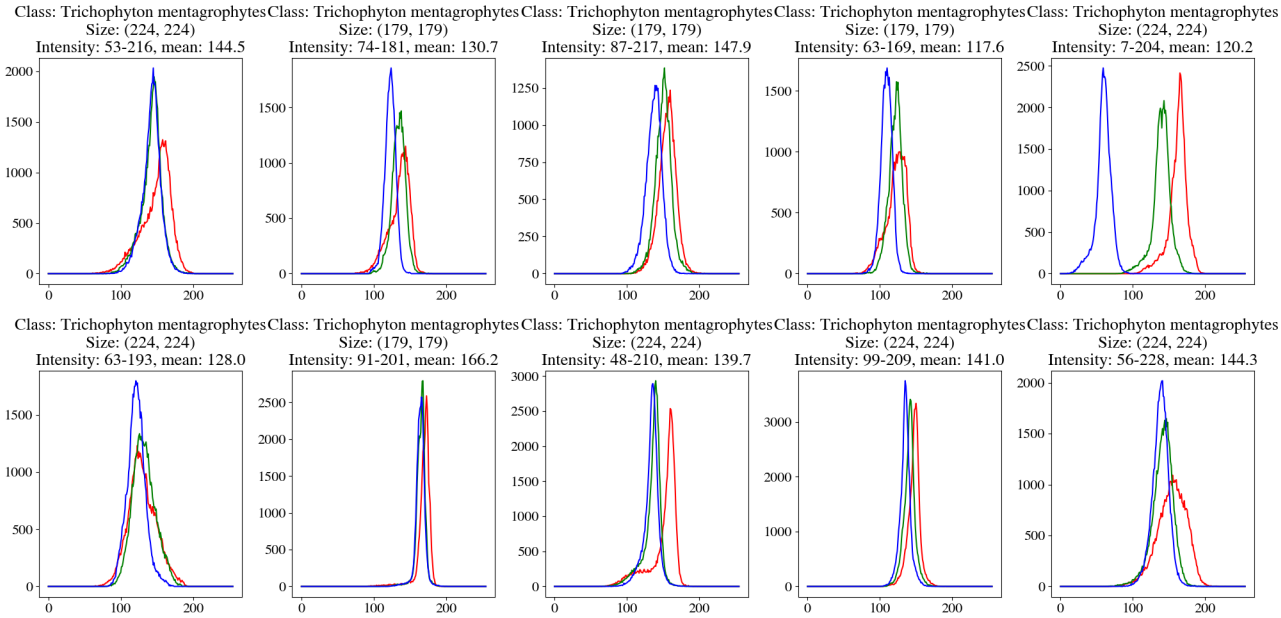


(b) Histograms

Figure 15: *Trichophyton mentagrophytes example images & RGB histograms*

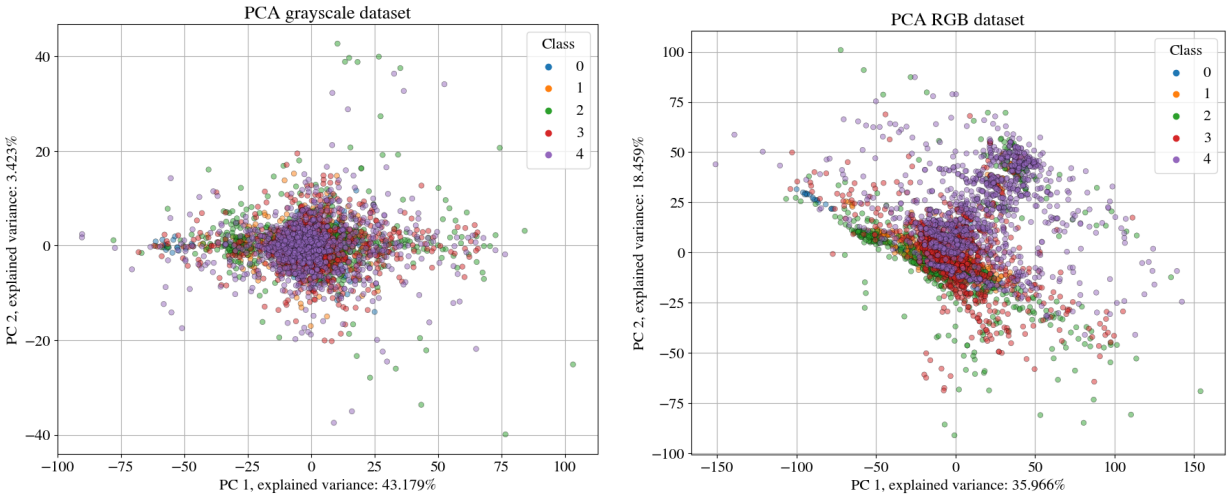


(a) Images



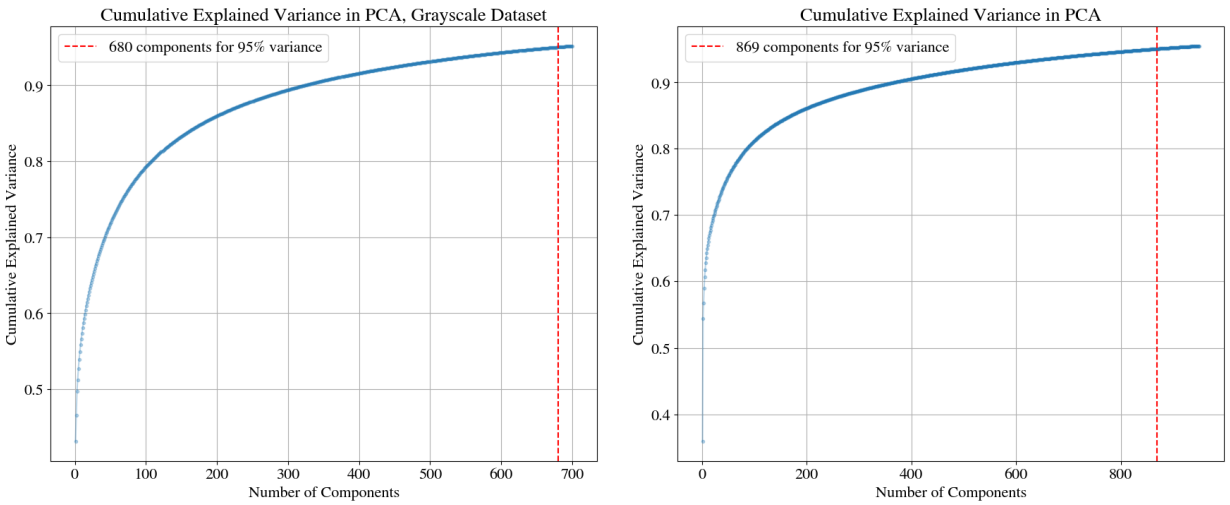
(b) Histograms

Figure 16: *Epidermophyton floccosum example images & RGB histograms*



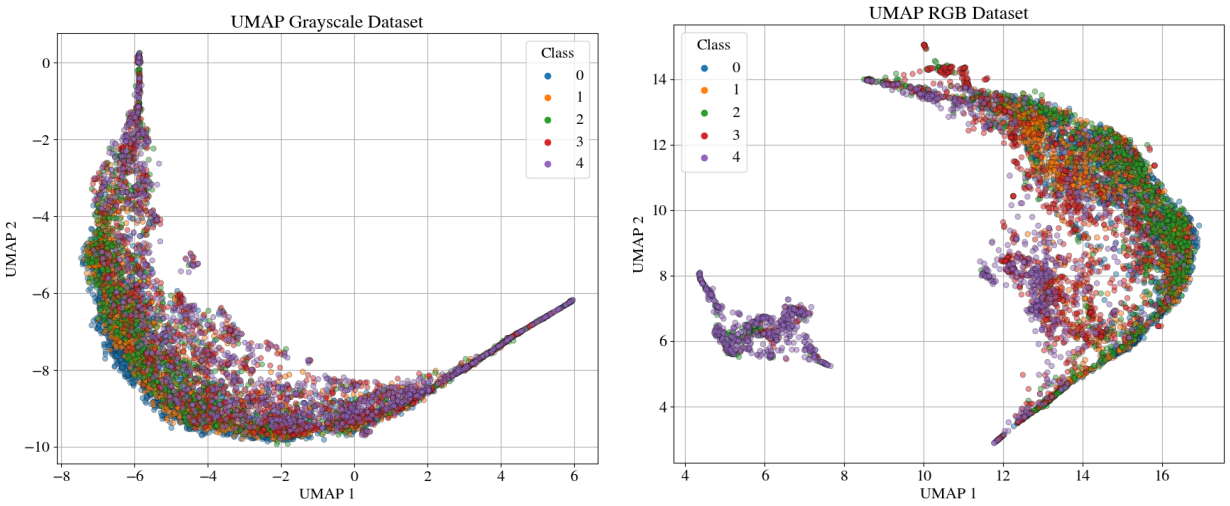
(a) Grayscale flattened dataset PCA (b) RGB flattened dataset PCA

Figure 17: *Visualization of the first two Principal Components*



(a) Grayscale flattened dataset PCA (b) RGB flattened dataset PCA

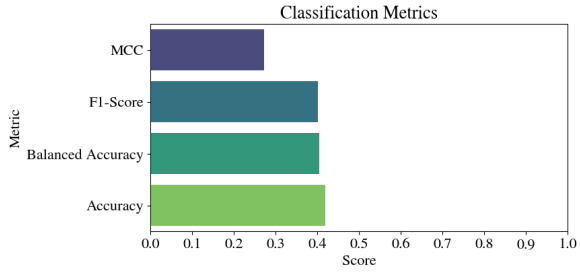
Figure 18: *Visualization of the first two Principal Components*



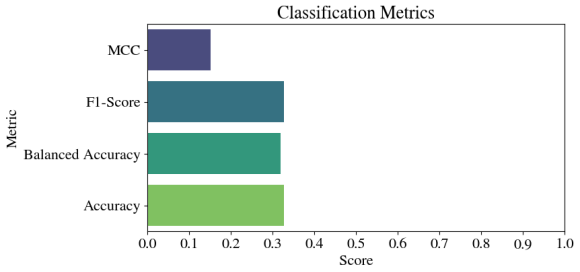
(a) Grayscale flattened dataset PCA

(b) RGB flattened dataset PCA

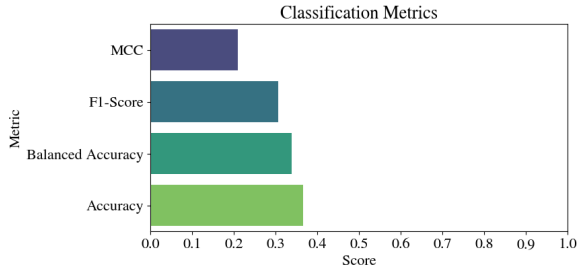
Figure 19: *Visualization of the first two Principal Components*



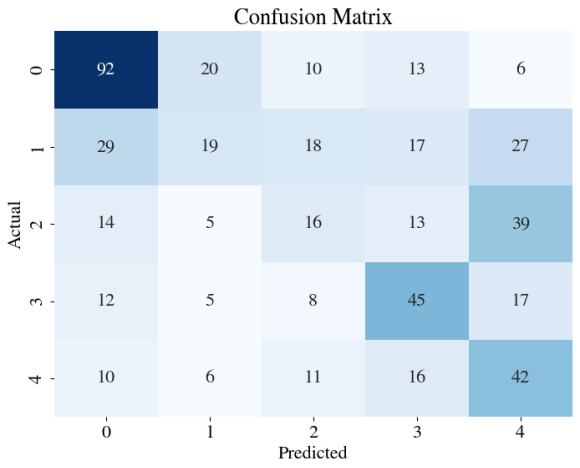
(a) Naive Bayes scores



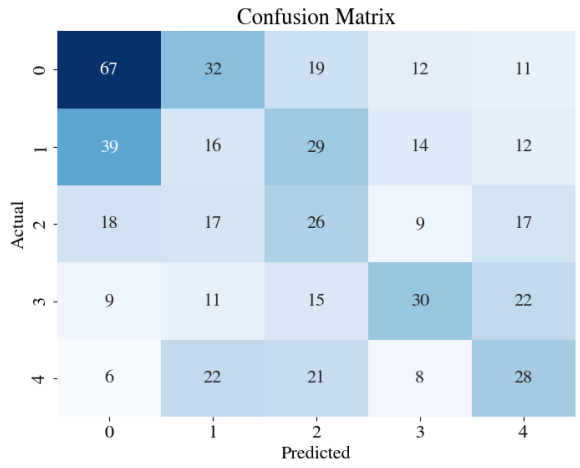
(b) Logistic Regression scores



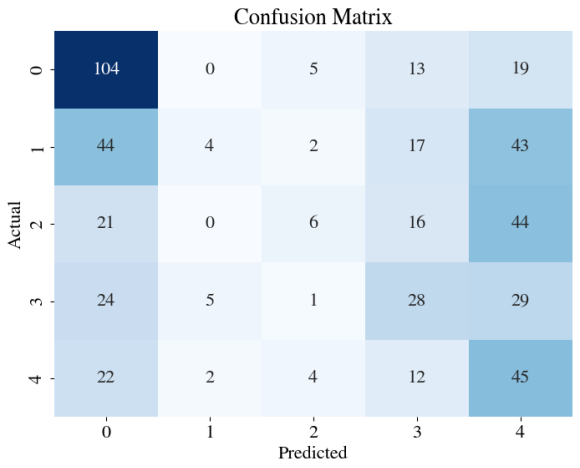
(c) Random Forest scores



(d) Naive Bayes confusion matrix

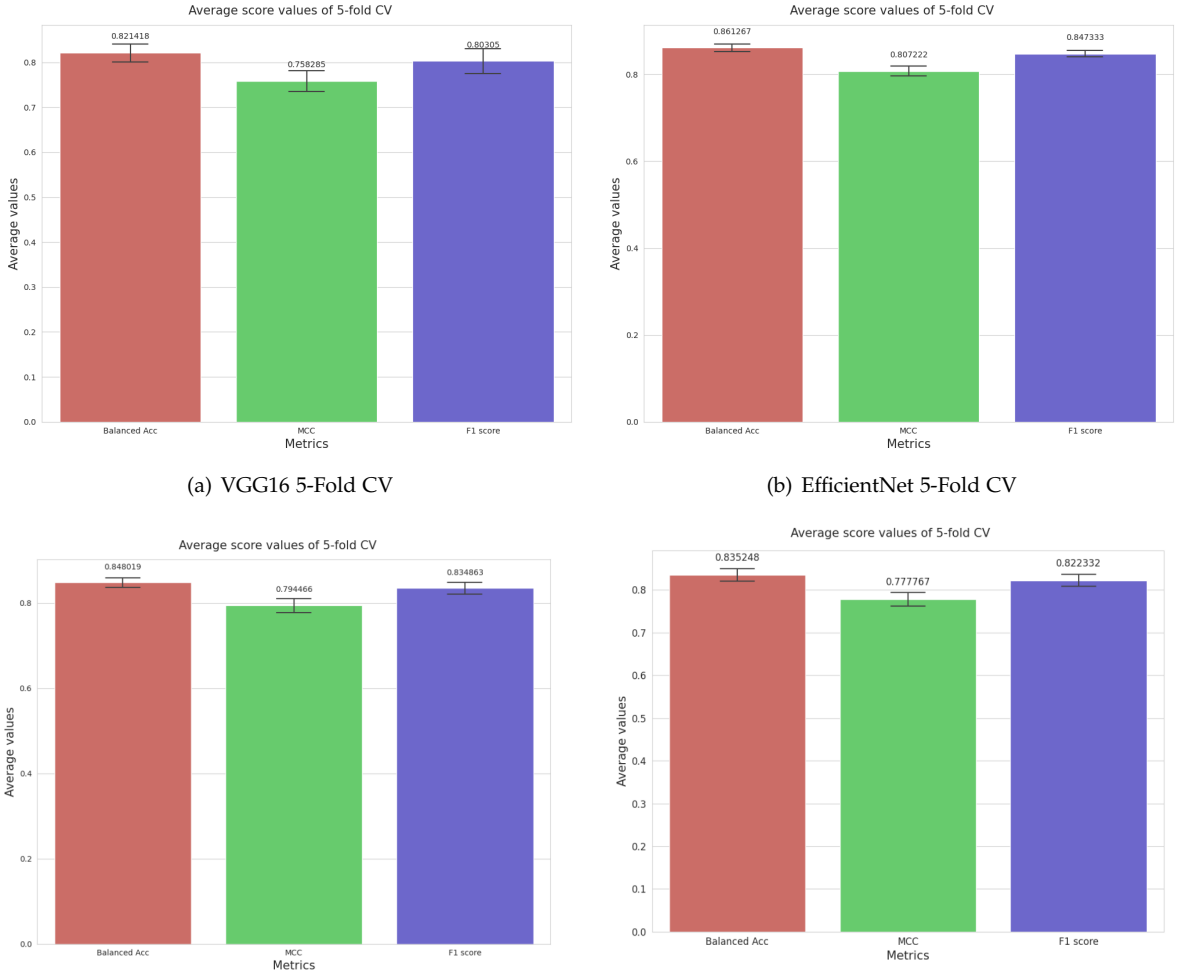


(e) Logistic Regression confusion matrix



(f) Random Forest confusion matrix

Figure 20: *Performance of classical ML algorithms on the HoG feature space*



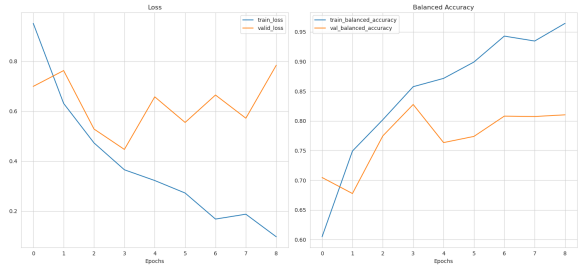
(a) VGG16 5-Fold CV

(b) EfficientNet 5-Fold CV

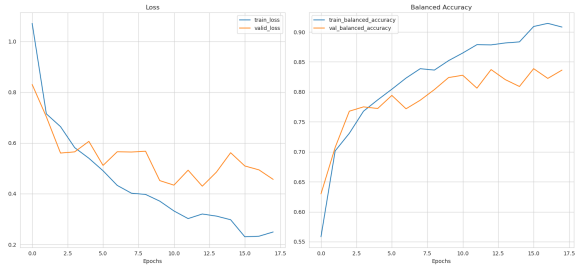
(c) ResNet 5-Fold CV

(d) ViT 5-Fold CV

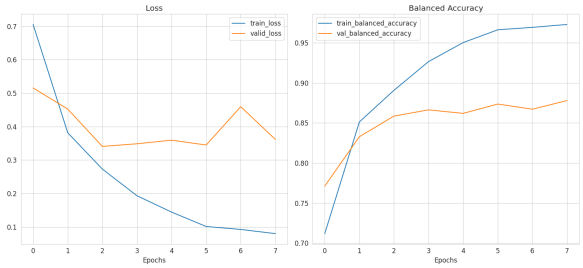
Figure 21: *5 fold cross validation average scores and standard deviations.*



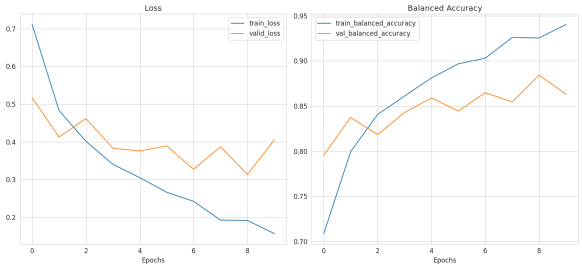
(a) VGG16 training curves without data augmentation



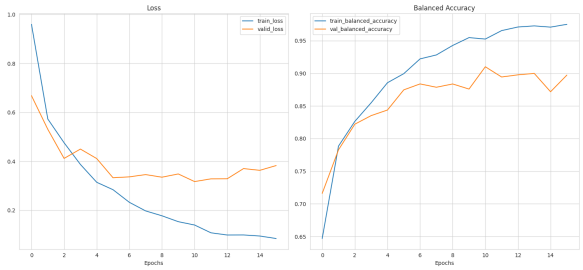
(b) VGG16 training curves with data augmentation



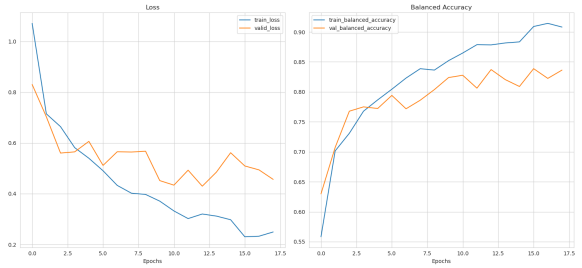
(c) ViT training curves without data augmentation



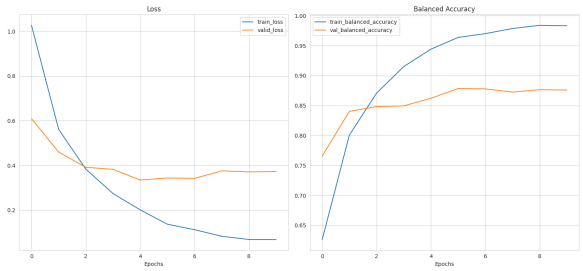
(d) ViT training curves with data augmentation



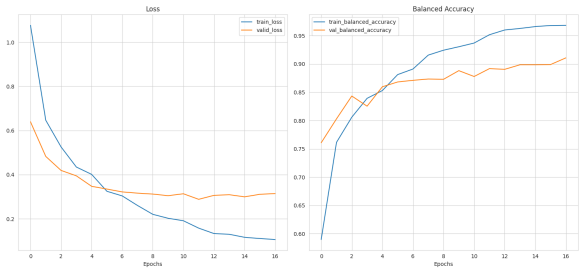
(e) ResNet50 training curves without data augmentation



(f) ResNet50 training curves with data augmentation

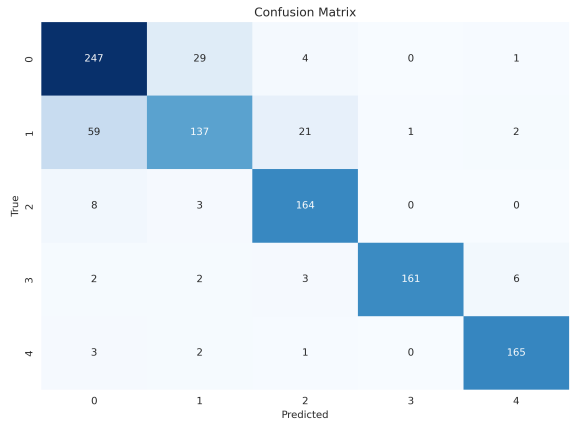


(g) EfficientNet training curves without data augmentation

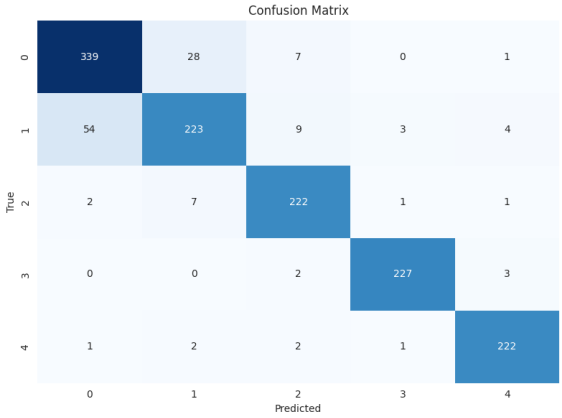


(h) EfficientNet training curves with data augmentation

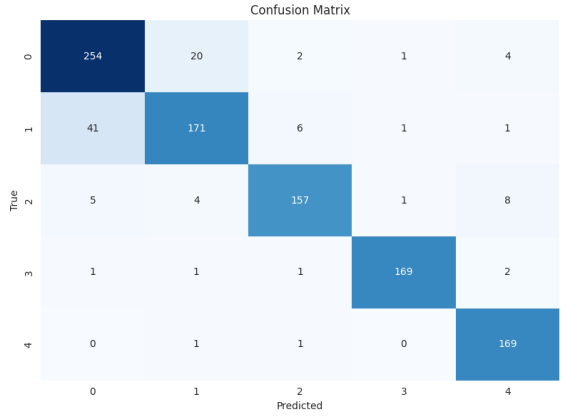
Figure 22: *Comparison of the models' training curves when trained with and without data augmentation.*



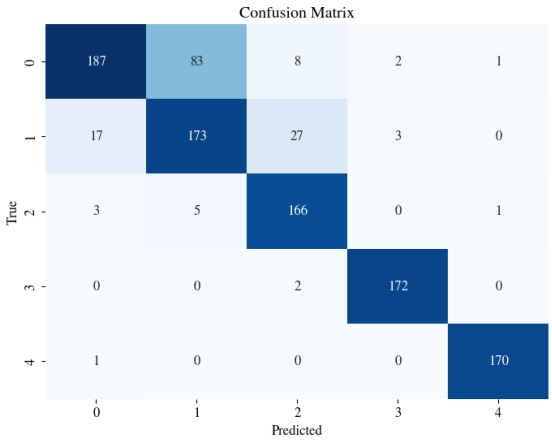
(a) VGG16 confusion matrix



(b) EfficientNetb0 confusion matrix

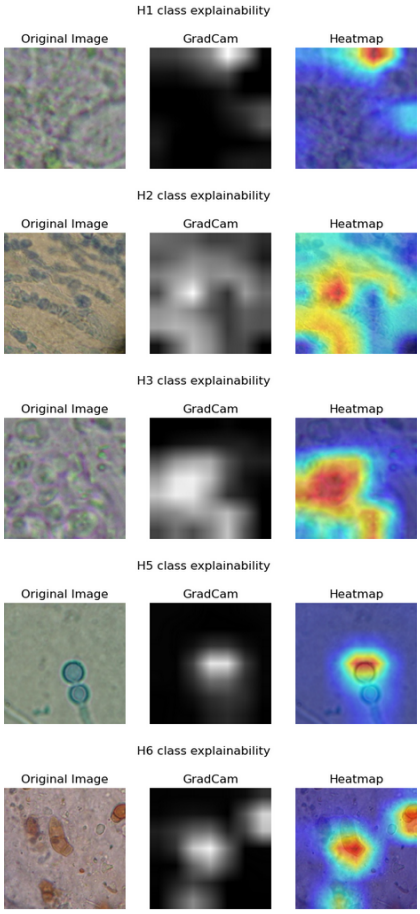


(c) ResNet50 confusion matrix

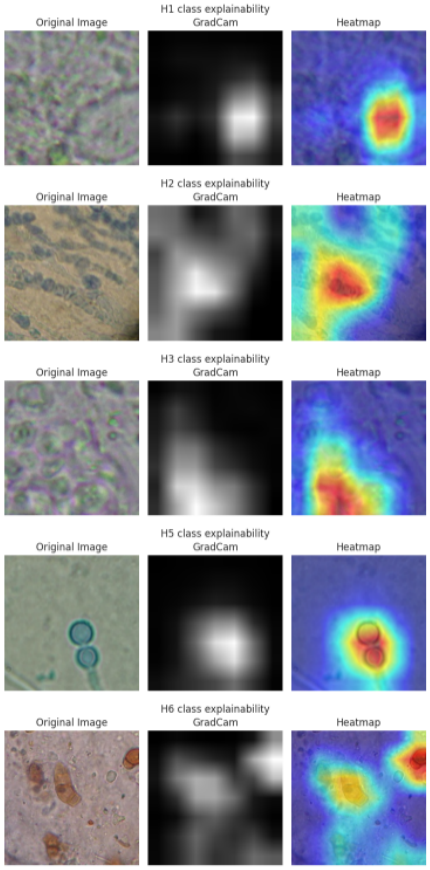


(d) ViT confusion matrix

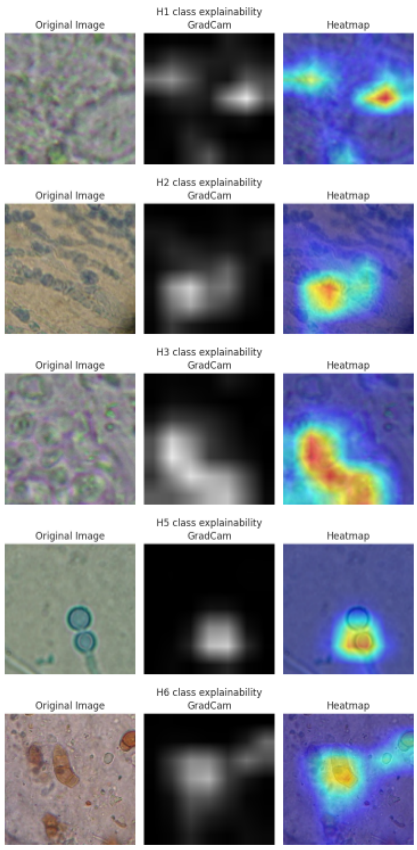
Figure 23: *Test set confusion matrices of the final models trained with data augmentation*



(a) VGG16 Grad-Cam Heatmaps

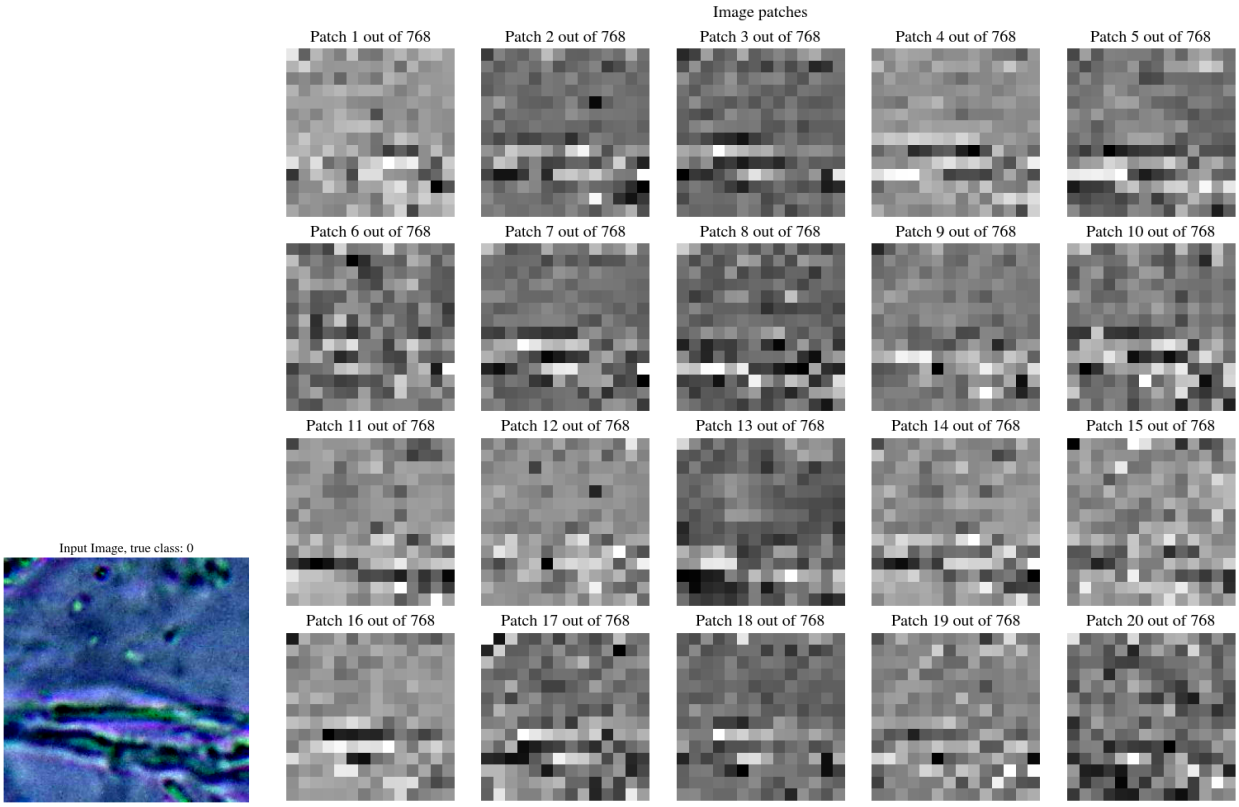


(b) EfficientNet Grad-Cam Heatmaps



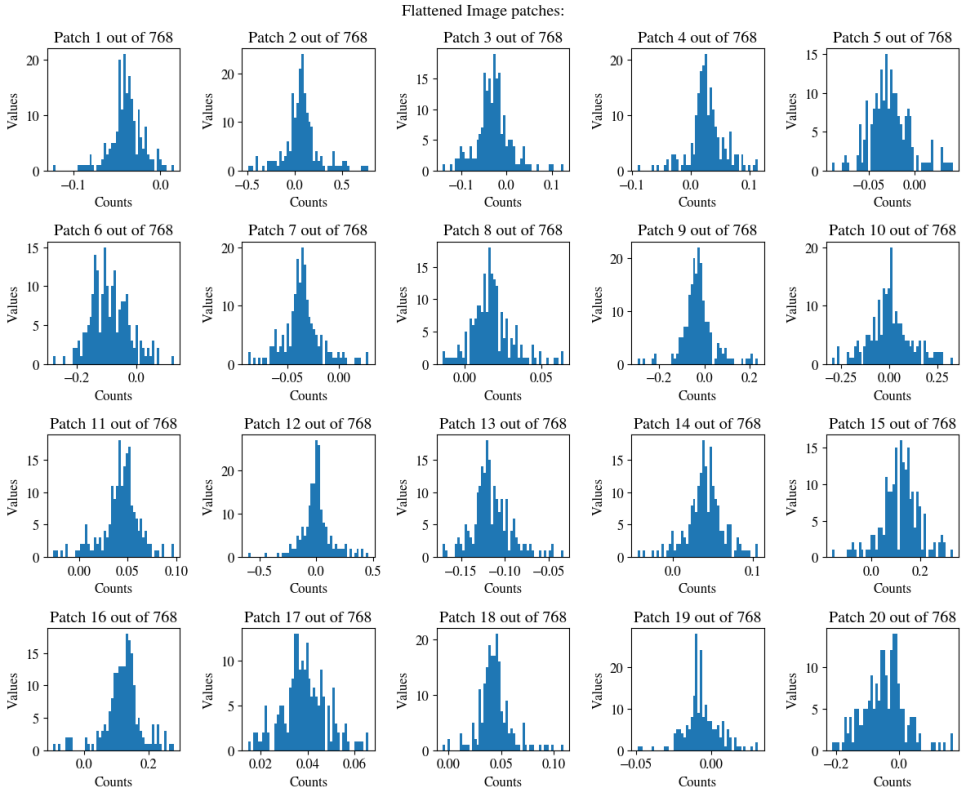
(c) ResNet50 Grad-Cam Heatmaps

Figure 24: *Visualization of the Grad-Cam Heatmaps produced on the CNN-based models.*



(a) Example input Image

(b) Image Patches



(c) Image Patches

Figure 25: *Visualization of the Grad-Cam Heatmaps produced on the CNN-based models.*

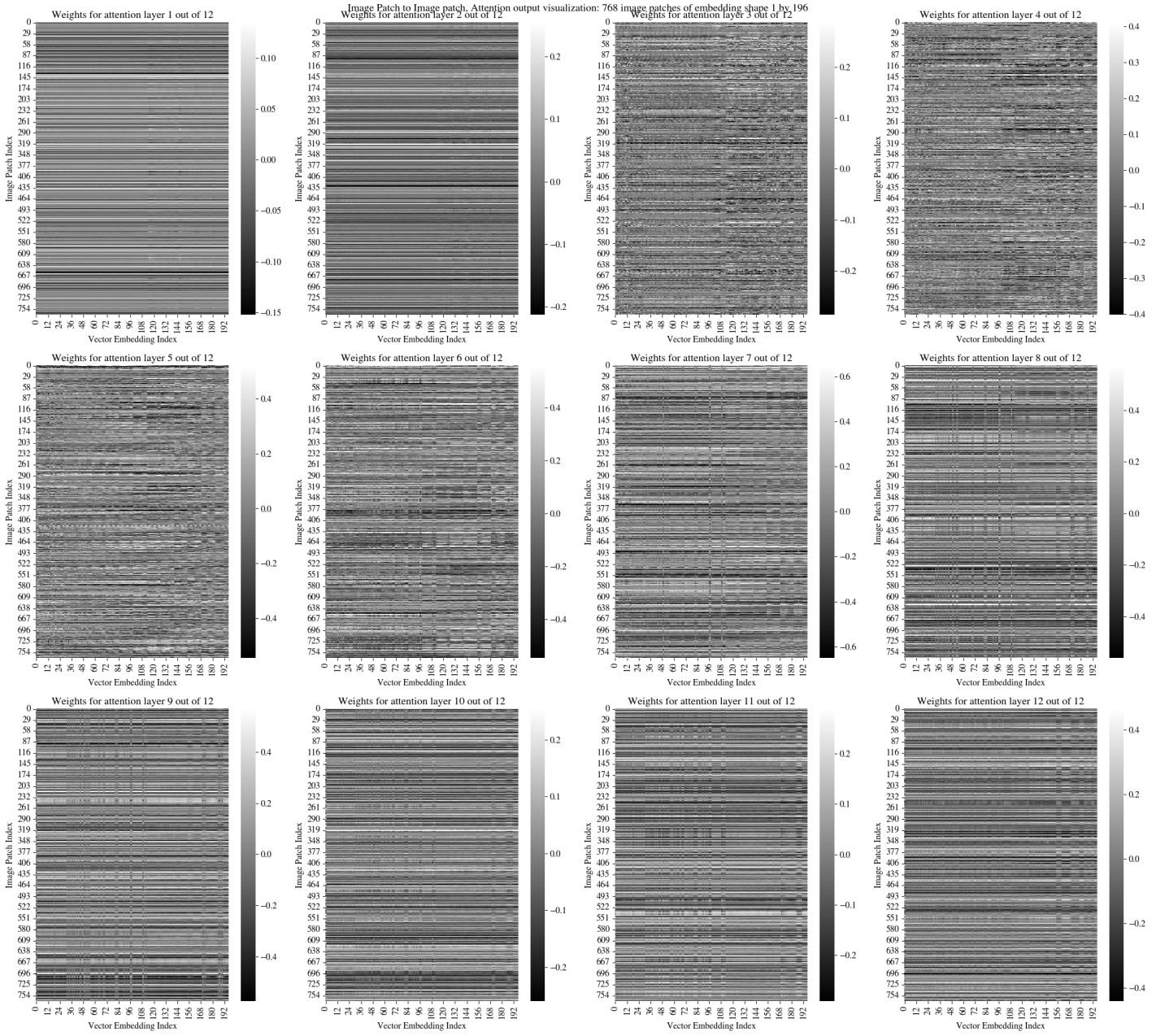


Figure 26: *Eaxmple Multi-Head-Attention layers output for ViT*