CSC462: Project Proposal

EfficientNet-Based Convolutional Neural Networks for Binary-class Image Classification

School of Computer Science, College of Computer and Information Sciences, KSU
Riyadh, Saudi Arabia

Hailah Alrumaih 441201376@student.ksu.edu.sa
Aljohara Aljubair 442200138@student.ksu.edu.sa
Waad Alahmed 442200402@student.ksu.edu.sa

1 Introduction

This work centers on the task of binary-class image classification, specifically focusing on the differentiation between buildings and forests. The practical implications of this task span various domains, and one illustrative example of its significance can be found in addressing the challenges of water resource management within the arid regions of Saudi Arabia. The persistent problem of water scarcity and resource management in such regions underscores the urgency for effective solutions. Accurate identification of areas bearing vegetation, such as forests or greenery around buildings, may serve as an essential indicator of potential water sources or resources for irrigation. As such, the classification of buildings and forests emerges as a pivotal component in addressing this multifaceted challenge, offering valuable insights into water source identification, land use planning, irrigation strategies, environmental protection, and disaster preparedness.

To address this classification task, Convolutional Neural Networks (CNNs) have emerged as a powerful tool, capable of making informed decisions even in the face of challenging climatic conditions. However, training deep neural networks from scratch can be a formidable endeavor, primarily due to limitations in computational resources, data availability, and time constraints. As a pragmatic alternative, fine-tuning a CNN that has been pre-trained on a vast, labeled dataset from a different application has become a preferred approach. In this context, we have selected EfficientNet-based CNNs for their high-capacity and efficiency, thus enhancing their suitability for this classification task.

In the subsequent sections, we will provide background, review related works, and introduce the dataset. We will then outline the proposed EfficientNet-based CNN approach and the methodology for model training and evaluation.

2 Background

EfficientNets [1] represent a family of CNNs architectures celebrated for their image classification capabilities. Within this family, there are several models, ranging from EfficientNet-B0 to B7, each crafted to strike an ideal equilibrium between model size, computational efficiency, and accuracy. EfficientNets emanates from two fundamental pillars: the model compound scaling and the EfficientNet architecture.

2.1 Compound Scaling

Before the advent of EfficientNets, traditional approaches to scaling CNNs involved adjusting a single dimension, such as depth (number of layers), width (number of channels), or image resolution (image size). However, EfficientNets employ a novel technique called compound scaling, which scales all three dimensions of a CNN architecture while preserving a delicate balance among them.

Compound scaling works by finding an optimal ratio between the width, depth, and resolution of a network to achieve improved accuracy and efficiency. This approach is motivated by the observation that uniformly adjusting all three dimensions using a set of predetermined scaling coefficients can lead to a more harmonious and efficient network architecture.

Compound scaling is illustrated in *Figure 1*, which shows how the width, depth, and resolution of an EfficientNet model are scaled.

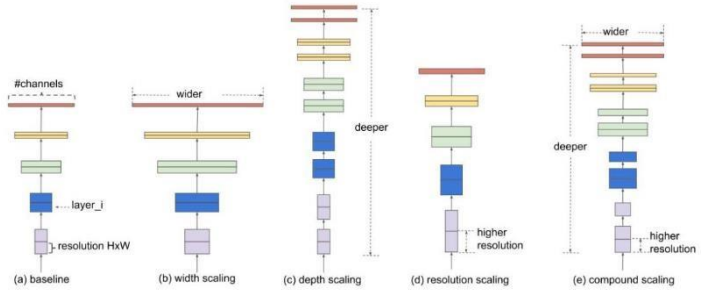


Figure 1. Compound scaling

2.2 EfficientNet architecture

The development of EfficientNet-B0, which serves as the baseline for the EfficientNet family of models, involves an innovative approach known as Neural Architecture Search (NAS). This method, similar to the one used in the MNasNet, leverages reinforcement learning principles to discover an optimal neural network architecture, illustrated in *Figure 2*. The process begins by defining a multi-objective search that optimizes for both model accuracy and Floating-Point Operations Per Second (FLOPS), a measure of computational efficiency. The optimization in this case focuses on FLOPS rather than latency, as it does not target specific hardware devices. The objective function can formally be defined as *Equation 1*.

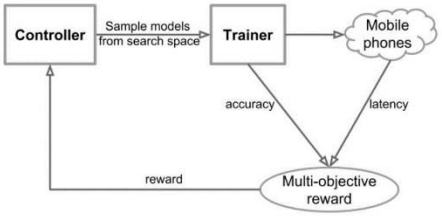


Figure 2. reinforcement learning principles

$$\underset{m}{\text{maximize}} \quad ACC(m) \times \left[\frac{LAT(m)}{T}\right]^{w}$$

where w is the weight factor defined as:

$$w = \begin{cases} \alpha, & \text{if } LAT(m) \leq T \\ \beta, & \text{otherwise} \end{cases}$$

Equation 1. Objective function

EfficientNet-B0, resulting from this architectural exploration, is defined in *Table 1*. One of its distinctive components is the MBConv layer, which resembles an inverted bottleneck block, illustrated in *Figure 3*. This layer incorporates a Depthwise Convolution to efficiently obtain output feature maps with less computations, illustrated in *Figure 4*. Furthermore, it includes squeeze-and-excitation optimization, enhancing the model's performance. This structural foundation, combined with the squeeze-and-excitation technique, forms the essence of EfficientNet-B0.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Table 1. EfficientNet-B0 architecture

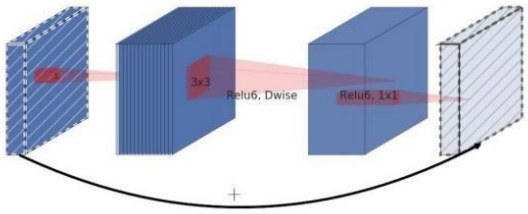


Figure 3. Inverted bottleneck block

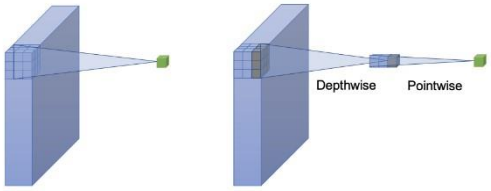


Figure 4. Standard convolution and depthwise separable convolution

The next step involves applying the Compound Scaling technique to scale the baseline network. This scaling process encompasses the dimensions of depth (d), width (w), and resolution (r) to create a family of EfficientNet models ranging from B1 to B7, illustrated in *Figure 5*.

```
EfficientNet Params          W          D          R
name: (channel_multiplier, depth_multiplier, resolution, dropout_rate)
'efficientnet-b0': (1.0, 1.0, 224, 0.2),
'efficientnet-b1': (1.0, 1.1, 240, 0.2),
'efficientnet-b2': (1.1, 1.2, 260, 0.3),
'efficientnet-b3': (1.2, 1.4, 300, 0.3),
'efficientnet-b4': (1.4, 1.8, 380, 0.4),
'efficientnet-b5': (1.6, 2.2, 456, 0.4),
'efficientnet-b6': (1.8, 2.6, 528, 0.5),
'efficientnet-b7': (2.0, 3.1, 600, 0.5),
'efficientnet-b8': (2.2, 3.6, 672, 0.5),
'efficientnet-l2': (4.3, 5.3, 800, 0.5),
```

Figure 5. EfficientNet models from B1 to B7

3 Related works

3.1 VGG-19 and resNet -50 pre-trained CNN model

In this paper [2], a study was conducted to evaluate the effectiveness of pre-trained deep CNNs models, including VGG-19 and ResNet-50, in comparison to training CNN model from scratch. The primary goal was to address the challenge of classifying chest X-ray images, particularly differentiating pneumonia cases. To mitigate overfitting, techniques like data augmentation and dropout regularization were employed. The study involved experiments using three publicly available chest X-ray datasets, specifically CheXpert from Stanford University, NIH, and chest X-ray images of pneumonia from Kaggle. Keras and TensorFlow, open-source Python libraries, were employed to train the CNNs for the classification of pneumonia cases.

The study encompassed a thorough performance evaluation that juxtaposed pretrained models VGG-19 and ResNet-50 with a CNN model named Iyke-Net, which was trained from scratch, as illustrated in *Figure 6*. VGG-19 demonstrated 97.3% accuracy, ResNet-50 reached 96.2%, while Iyke-Net achieved an accuracy of 93.60%. These findings underscore the advantage of pre-trained models, underscoring their high performance in image classification tasks in comparison to models built from the beginning.

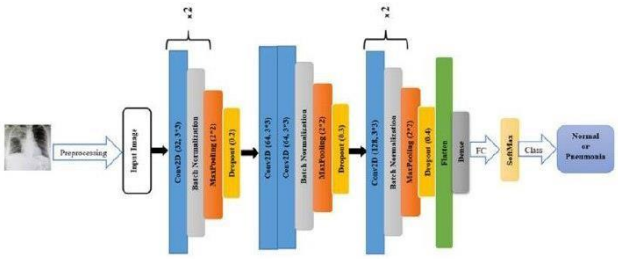


Figure 6. Block diagram of lykeNet

3.2 resNet -18 and resNet -50 pre-trained CNN model

In this study [3], the research focused on using the ResNet architecture to classify colorectal cancer. Devvi et al. [3] conducted experiments to determine how ResNet-18 and ResNet-50 models performed when distinguishing between benign and malignant colorectal cancer. They utilized the Warwick-QU dataset, consisting of 165 images, 74 benign and 91 malignant tumors.

Following the preprocessing steps, which involved grayscale conversion, CLAHEbased contrast enhancement, and resizing to a 224 x 224 grid, the researchers employed ResNet models for image classification. ResNet-18 and ResNet-50, denoted by the numbers 18 and 50, respectively, represent the number of layers within each architecture. Both models incorporated convolution layers, batch normalization, activation layers with ReLU activation, and pooling layers. Additionally, residual blocks were integrated within the ResNet architecture, dividing the convolution layers into five stages.

The results demonstrated that ResNet-50 outperformed ResNet-18 in terms of accuracy, sensitivity, and specificity across different test data sizes. The highest performance was achieved with 20% test set, exhibiting a classification accuracy of 88%, sensitivity of 93%, and specificity of 83% across these datasets.

3.3 Inception-ResNet-v2 pre-trained CNN model

In a study [4], researchers utilized a Deep CNN model called the pre-trained Inception-ResNet-v2 (IR) for domain adaptation in image recognition. Their methodology involved extracting features using the IR model and then employing the Manifold Embedded Distribution Alignment (MEDA) technique to classify these features, as demonstrated in *Figure 7*. The combined approach, referred to as the MDAIR model, incorporates both the IR model for feature extraction and MEDA for classification.

The research involved four distinct datasets: Office-10, Office-31, Caltech-10, and Office-Home. Office-10, comprising 1,410 samples; Caltech-10, with 1,123 samples; Office-31, totaling 1,330 samples; and Office-Home, the largest among them, containing 15,588 samples. The features extracted from the Inception-ResNet model for all these datasets consisted of 1,000 features, categorized into 10 classes for the first two datasets, 31 classes for the third dataset, and 65 classes for the fourth dataset.

The research findings demonstrated the superiority of MDAIR model compared to other baseline models. Specifically, for the first and second datasets, the average accuracy reached 96.7%, outperforming other models by 4.8%. In the third dataset, the average accuracy achieved 89.8%, exceeding other models by 5.5%. Lastly, for the fourth and largest dataset, the average accuracy was 72.8%, surpassing other models by 10%.

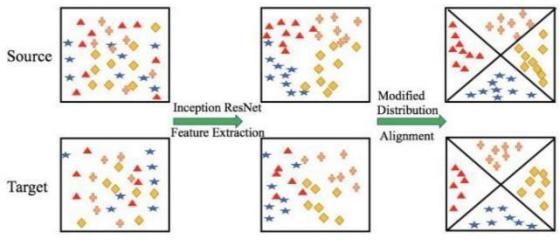


Figure 7. The scheme of MDAIR model

4 Dataset

The dataset's primary objective is to classify images into two distinct categories: "Building" and "Forest." Sourced from Kaggle [5], the dataset comprises a total of four files, divided into two separate sets for training and testing. In its entirety, the dataset includes 5,354 images, with an equal distribution between the "Building" and "Forest" categories. Within the training set, there are 4,443 images, while the testing set contains 911 images, representing 17% of the entire dataset.

4.1 Building category

The training set includes 2,190 images, with 437 images in the testing set. *Figure 8* provides a visual representation of sample images from this category.

Figure 8. Sample images from Building category

4.2 Forest category

The training set contains 2,253 images, and the testing set has 474 images. *Figure 9* showcases a sample image from this category.

Figure 9. Sample images from Forest category

5 Proposed Approach and Methodology

EfficientNet-B0 has been chosen as the architectural model for our project, superseding the initial selection of EfficientNet-B7. This decision is rooted in the necessity to address computational resource constraints, specifically the challenge of memory exhaustion associated with the more sizable EfficientNet-B7 model. As the most compact variant within the EfficientNet series, EfficientNet-B0 stands out for its memory efficiency and reduced computational intensity. These attributes render it particularly suitable for deployment in scenarios characterized by limited GPU memory capacity.

In terms of performance, EfficientNet-B0 exhibits no significant compromise, despite its reduced scale, especially in the context of our designated task - the classification of images into two distinct categories: buildings and forests. This specific task does not require the heightened complexity and extended capacity offered by the B7 model.

The adoption of EfficientNet-B0 within the PyTorch framework for our project is driven by the model's resource efficiency and PyTorch's flexibility and ease of use. This combination allows us to efficiently manage our image classification task, leveraging PyTorch's robust functionalities for streamlined model training and optimal performance. This approach effectively addresses our project needs while avoiding the high resource demands of more complex models.

Note: The transition to EfficientNet-B0 was necessitated by the emergence of memory overflow issues during the use of EfficientNet-B7, a recurrent complication when handling extensive models on hardware with limited capabilities. The adoption of EfficientNet-B0, with its lower complexity and reduced memory footprint, provides a viable solution for our project requirements, ensuring judicious use of resources while maintaining the efficacy of the model in accurately categorizing images.

In the following sections, we discuss the steps for building and training the EfficientNet-B0 model. Details of the code will be provided at the end of the document to maintain a clear narrative flow and focus on the methodology first.

5.1 Data Preparation and Analysis: Training and Testing Dataset Split, Visualization, and Class Distribution

- **Import necessary library:**

The project begins by importing key libraries essential for machine learning development. torch serves as the foundational library for PyTorch, providing functionalities for neural network construction and training. *torchvision* offers tools for image data handling, including datasets and model architectures. *torchvision.transforms* is used for image preprocessing and augmentation. *ImageFolder from torchvision.datasets* loads datasets where each class is in a separate folder. The *timm* library gives access to pre-trained models like EfficientNet. *torch.utils.data*, *torch.nn*, and *torch.optim* are PyTorch modules for data handling, neural network components, and optimization methods. The Python Imaging Library *PIL.Image* manages image file operations. *IPython.display* is used for media display in Jupyter notebooks. *os* and *collections.Counter* are standard Python libraries for system operations and object counting. Finally, *numpy* and *matplotlib.pyplot* are essential for numerical computations and data visualization.

- **Define Preprocessing Techniques:**

This process involves a series of transformations to standardize and enhance the data, ensuring it is in an optimal form for feeding into the neural network. The transformations include:

Resizing Images: Each image in the dataset is resized to a uniform dimension of 224x224 pixels. This uniformity is crucial as it ensures that all images fed into the model have a consistent size, which is a prerequisite for most convolutional neural network architectures.

Converting to Tensors: The images are converted from their native format (like JPEG or PNG) into PyTorch tensors. Tensors are a fundamental data structure in PyTorch and are used to encode the features and labels in a format that is suitable for machine learning models.

Random Horizontal Flips: This transformation randomly flips images horizontally. It's a form of data augmentation that helps the model generalize better by creating a more diverse set of training examples from the existing data. By seeing flipped versions of images, the model learns to recognize objects regardless of their orientation in the image.

Normalization: The pixel values of the images are normalized. Typically, this involves scaling the pixel intensity values to have a certain mean and standard deviation. Normalization helps in speeding up the convergence of the model during training and reduces the sensitivity to the scale of features.

- **Dataset Loading: Splitting into Training and Testing Sets**

Specifying Class Names: It's essential to explicitly specify the class names 'buildings' and 'forests' for the model to accurately process and classify the images. These designated class labels allow the model to distinguish and categorize the input images into their respective categories effectively.

Loading Datasets: Training and testing data are loaded from designated directories using the ImageFolder class, applying predefined image transformations.

Creating Data Loaders: DataLoader instances for both sets manage data batching (with a batch size of 16) and shuffling, key for effective model training and evaluation. This setup ensures optimal memory usage and learning efficiency.

- **Data Exploration and Analysis:**

Post-preprocessing, the dataset undergoes exploration and analysis:

Sample Image Visualization: Images from training and testing sets are showcased to assess characteristics like color, composition, and themes pertinent to 'buildings' and 'forests'.

Average Color Intensity: This analysis calculates mean pixel intensities across images, shedding light on color distribution and potential dataset biases.

Image Shape Diversity: This analysis involves examining the variety of image dimensions in the dataset. It helps in confirming that the preprocessing steps, particularly image resizing, are appropriately configured for the model.

Class Distribution: Evaluating the balance of 'building' and 'forest' images is essential to avoid training biases towards any one class.

5.2 Modify and Fine-Tune the EfficientNetB0 Model

In this step, we focus on customizing and optimizing the EfficientNet-B0 model for our specific classification task, which involves distinguishing between 'buildings' and 'forests'. The modifications and fine-tuning procedures include:

Model Initialization: EfficientNet-B0 is loaded without pretrained weights using the timm library. This version of EfficientNet is selected for its balance between computational efficiency and model performance.

Fine-tuning the EfficientNet-B0 model for binary classification: it involves replacing the original classifier with a new linear layer, outputting two features for 'buildings' and 'forests'. Simultaneously, the model adopts the SiLU (Swish) activation function (`SiLU(inplace=True)`), blending linear and sigmoid characteristics to effectively learn complex patterns. The `inplace=True` setting optimizes memory use, essential for efficient resource management. This fine-tuning ensures the model accurately distinguishes between the two classes, capturing their distinct features.

The concept of using the EfficientNet-B0 model for binary image classification, complemented by a fine-tuning process, can be effectively summarized by the following diagram referred to as *Figure 10*. This illustrates the streamlined journey from image input through the fine-tuning stage to the final classification output as either 'Building' or 'Forest.

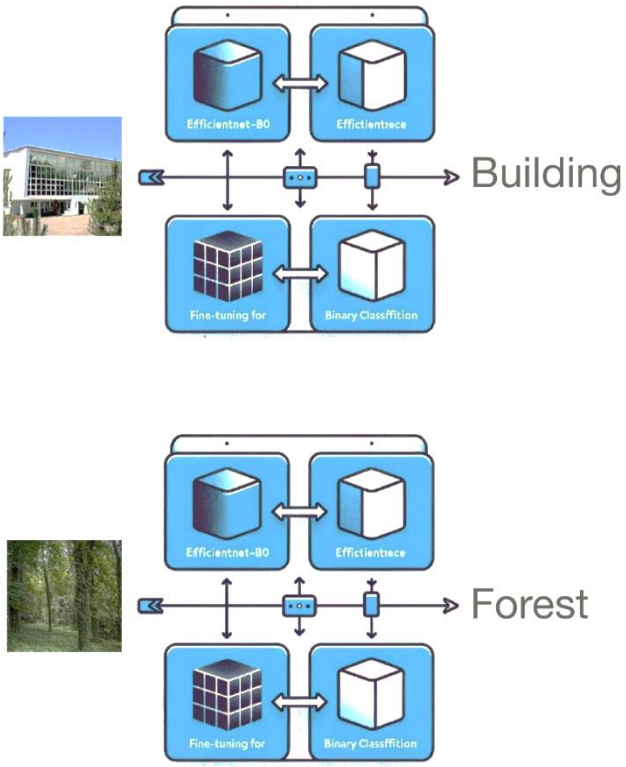


Figure 10. Model architecture

5.3 Model Training

Throughout the training process, our model's parameters are adjusted based on feedback derived from a loss function. To identify the optimal training configurations, we conducted five experiments employing a range of hyperparameters, as outlined in *Table 2*. These experiments investigated the influence of various settings on the model's performance. The findings gleaned from these experiments will steer the optimization of our model, culminating in enhanced accuracy and generalization capability in classifying images between buildings and forests.

Experimen	***1***	***2***	***3***	***4***	***5***
Learning Rate	0.01	0.01	0.1	0.01	0.001
Loss function	Cross Entropy				
optimizer	SGD	SGD	Adam	AdamW	SGD
L2 regularization (weight_decay)	0.0001	0.005	Not used	Not used	0.005
Number of layers	One fully connected layer + EffentNetB0 Layers				
Neurons in Hidden Layers	No hidden layer in fully connected layer + Neurons in hidden layers for EffentNetB0 model				
Activation Function	Sigmoid function (SiLU)				
gamma	0.1	0.1	0.4	0.1	0.1
Number of Epochs	6	6	6	6	6
Batch Size	8	8	8	8	8
Step size	30	30	30	70	30
Momentum	0.9	0.7	Not used	Not used	0.8

Table 2. Five experiments employing a range of hyperparameters

- **Experiment 1: Baseline with SGD**

Description: Establishes a baseline using the Mini-Batch Gradient Descent (SGD) optimizer with a learning rate of 0.01, weight decay for L2 regularization set at 0.0001, and a momentum of 0.9. The model will be trained for 6 epochs with a batch size of 8 and a step size of 30 epochs for the learning rate scheduler. **Purpose:** To determine the foundational performance metrics for subsequent comparisons.

- **Experiment 2: Higher L2 regularization**

Description: Adopts the SGD optimizer with a higher weight decay of 0.005 for L2 regularization and momentum 0.7. The learning rate, and other training parameters are kept consistent with the first experiment. **Purpose**: To determine if with a higher weight decay of 0.005 for L2 regularization and smaller momentum leads to better generalization and higher accuracy in the classification task.

- **Experiment 3: Without Regularization and momentum with Adam optimizer**

Description: Implements the Adam optimizer with increased gamma to be 0.4 and the learning rate to be 0.1. Training parameters such as epochs, batch size, and step size remain unchanged. **Purpose:** To observe the adaptability of the Adam optimizer with model.

- **Experiment 4 Without Regularization and momentum with AdamW**

Description: Utilizes the AdamW optimizer. The experiment keeps the gamma at 0.1, and other training parameters consistent with previous experiments, only decrease learning rate to be 0.01. **Purpose:** To assess the impact of AdamW's built-in weight decay mechanism in the absence of additional L2 regularization.

- **Experiment 5: Different Loss Function and Optimizer**

Description: SGD optimizer. It employs a learning rate of 0.001 with a momentum 0.8 and weight decay of 0.005 for L2 regularization. **Purpose:** To investigate the efficacy of the SGD optimizer in conjunction with adjusted momentum and weight decay settings. This setup aims to explore the balance between learning rate, regularization, and momentum to optimize model training and performance.

5.4 Model Evaluation

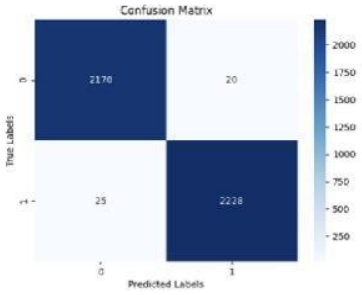
In the model evaluation phase, we rigorously assessed the performance of our EfficientNet-B0 model across five distinct experiments. Each experiment was subjected to meticulous evaluation using various metrics. The primary measure used during training was the loss function, which guided the optimization process. Additionally, we determined the model's accuracy on both the training and testing datasets. For a more detailed analysis, confusion matrices were generated to understand the model's performance in differentiating between the two classes, buildings and forests, on the training set. This assessment provided a comprehensive view of the model's learning efficiency and generalization capabilities.

The results of these evaluations, encompassing training accuracy, testing accuracy, and confusion matrices, are comprehensively illustrated in *Figures 11*. These figures provide a visual representation of the model's performance metrics.

Experiment 1:

Accuracy of the model on the train images: 98.98717083051991%

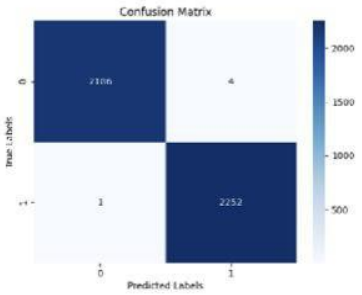
Accuracy of the model on the test images: 98.13391877058177%



Experiment 2:

Accuracy of the model on the train images: 99.88746342561332%

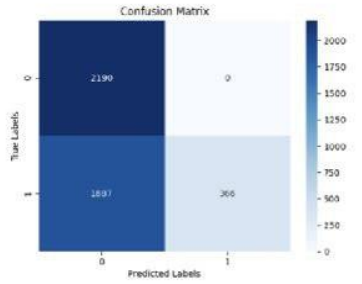
Accuracy of the model on the test images: 98.35345773874863%



Experiment 3:

Accuracy of the model on the train images: 57.528696826468604%

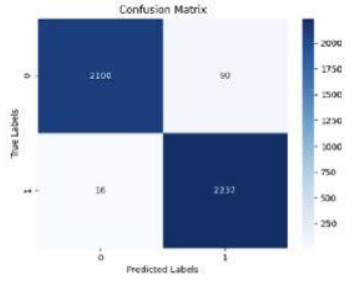
Accuracy of the model on the test images: 54.665203073545555%



Experiment 4:

Accuracy of the model on the train images: 97.61422462300247%

Accuracy of the model on the test images: 98.02414928649836%



Experiment 5:

Accuracy of the model on the train images: 98.35696601395453%

Accuracy of the model on the test images: 98.79253567508233%

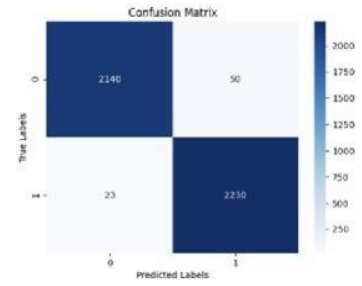


Figure 11. Training accuracy, testing accuracy, and confusion matrices

5.5 Predicting Unseen Images

To predict the class of unseen images, we have established a procedure that starts by sourcing images from external sources, ensuring they are distinct from those used during the training and testing phases. Each image undergoes a series of preprocessing steps, which are consistent with the transformations applied to the training dataset to maintain model compatibility.

Once preprocessed, each image is treated as a batch, despite it being a single image, to comply with the input requirements of the PyTorch framework. The model is then set to evaluation mode, which deactivates dropout layers and batch normalization, ensuring consistent predictions. This is crucial as the training mode would yield varied results due to the stochastic nature of these layers.

The images are then loaded onto the CPU for inference. In scenarios where GPU resources are scarce or unavailable, CPU-based inference provides a more accessible, albeit slower, alternative for image classification tasks.

The prediction is carried out without the need to track gradients, as we do not require backpropagation during the inference stage. The output from the model provides the class indices with the highest predicted probabilities. We then map these indices to their respective class names, which, in the context of our project, are 'Building' or 'Forest'. The result is a user-friendly class name output that indicates the model's prediction for each unseen image.

This process has been tested on a small subset of external images, and the results have been encouraging, with the model accurately predicting the class labels. This demonstrates the model's capability to generalize from its training on the provided dataset to new, unseen data.

The results of prediction are effectively illustrated in *Figure 12*, which showcases an example of the model's capability to accurately classify images based on the learned characteristics of buildings and forests.

Figure 12. Predict the class of unseen images

6 Discussion: Analysis of Model Performance and Experimentation Insights

In the pursuit of optimizing the EfficientNet-B0 model for binary classification of images into 'buildings' and 'forests', five distinct experiments were conducted. Each experiment was meticulously designed to investigate the impact of various hyperparameters on model accuracy and generalization. Here, we present a detailed analysis and comparison of these experiments, culminating in the identification of the most effective configuration.

Baseline with SGD (Experiment 1): This initial experiment established a foundational performance benchmark. While reliable, it highlighted the potential for enhancements through refined hyperparameter tuning.

Enhanced L2 Regularization (Experiment 2): Increasing the weight decay aimed to improve generalization. Although it slightly mitigated overfitting, the overall effect on accuracy was marginal.

Adam Optimizer (Experiment 3): Notably, this experiment's performance was below expectations, with an accuracy of 57%. The increase in the gamma value to 0.4 potentially contributed to this reduced effectiveness.

AdamW Optimizer (Experiment 4): Leveraging AdamW's built-in weight decay mechanism, this setup demonstrated promise, suggesting the efficacy of AdamW's regularization approach.

SGD with Adjusted Parameters (Experiment 5): This configuration excelled, outshining other setups with an accuracy of 98.79%. The combination of a learning rate of 0.001 and a weight decay of 0.005 proved optimal, balancing rapid convergence with the prevention of overfitting. The choice of cross entropy as the loss function resonated well with our binary classification task.

Best Performing Experiment: Experiment 5 stands out as the most effective configuration, marked by several key elements:

- Optimization Strategy: Utilizing Mini-Batch Gradient Descent (SGD), this approach efficiently navigated the dataset's error landscape.
- Optimal Learning Rate and Regularization: A learning rate of 0.001, coupled with a weight decay of 0.005, effectively balanced quick convergence and the minimization of overfitting.
- Loss Function Alignment: The use of cross entropy was in sync with the binary nature of the classification task, providing an effective mechanism for model updates.

The experimentation phase was crucial in elucidating the intricate interplay of hyperparameters in deep learning models. It highlighted the significance of comprehensive hyperparameter tuning, where each element is pivotal in guiding the model towards enhanced accuracy and robust generalization. Experiment 5, as the best-performing setup, establishes a benchmark for future endeavors in similar classification tasks, underscoring the value of thoughtful and informed hyperparameter optimization in deep learning.

7 Project strengths

Our project boasts several notable strengths. Firstly, the EfficientNet-B0 model at its core offers an excellent balance between performance and computational efficiency, making it ideal for environments with limited resources. It has been optimized to provide high accuracy despite its relatively small size and computational footprint. Additionally, the fine-tuning process has been meticulously crafted to adapt the model to our specific task of binary classification of buildings and forests, which has resulted in impressive classification accuracy on both the training and unseen external images.

8 Limitations

One of the project's primary limitations is the model's dependency on the quality and diversity of the training data, with limited variation leading to potential biases or overfitting. While EfficientNet-B0 is optimized for efficiency, there are specialized architectures that could yield better performance, but may require more computational resources. Additionally, throughout the project, we encountered significant constraints related to hardware capabilities, particularly with memory limitations. This not only restricted our choice of model to EfficientNet-B0 over the larger B7 variant but also imposed constraints on the batch sizes and image resolutions that could be used during training. These memory issues highlight the need for careful management of computational resources and potentially further optimization of the model and training process to accommodate hardware limitations.

9 Future Work

Looking ahead, the project can be expanded in several ways. Exploring more advanced data augmentation techniques could further improve the model's robustness and generalization. Additionally, experimenting with different architectures or newer versions of EfficientNet could yield even more accurate models. Implementing a more comprehensive hyperparameter tuning process, potentially through automated machine learning pipelines, could also refine model performance.

10 Conclusion

In conclusion, this project has successfully demonstrated the feasibility of using convolutional neural networks, specifically the EfficientNet-B0 model, for the task of binary image classification. Through careful preprocessing and fine-tuning, the model has achieved high accuracy, proving its effectiveness. While there are limitations and scope for further work, the current outcomes are promising and lay a solid foundation for future exploration and development in the field of image classification.

Work distribution

Student Name	*Task*
Hailah Alrumaih	✓ Introduction ✓ Related works ✓ Dataset ✓ Project strengths - Limitations ✓ Future Work
Aljohara Aljubair	✓ Proposed Approach and Methodology ✓ Discussion: Analysis of Model Performance and Experimentation Insights
Waad Alahmed	✓ Background ✓ Conclusion ✓ Code implementation

References

[1] Arora, A. (2020) *EfficientNet*, *index*. Available at: https://amaarora.github.io/posts/2020-08-13-efficientnet.html (Accessed: 05 November 2023).

[2] Ikechukwu, V., Murali, S., Deepu, R., & Shivamurthy, R. C. (2021). ResNet-50 vs VGG-19 vs training from scratch: A comparative analysis of the segmentation and classification of Pneumonia from chest X-ray images. In 2021 IEEE International Geoscience and Remote Sensing Symposium (IGARSS) (pp. 2070-2073). IEEE.

[3] Sarwinda, D., Paradisa, R. H., Bustamam, A., & Anggia, P. (2022). Deep Learning in Image Classification using Residual Network (ResNet) Variants for Detection of Colorectal Cancer. In 2022 9th International Conference on Information Technology and Applications (ICITA) (pp. 1-5). IEEE.

[4] Zhang, Y., & Davison, B. D. (2019, July). Modified distribution alignment for domain adaptation with pre-trained inception resnet. In 2019 IEEE International Conference on Computer Vision (ICCV) (pp. 2633-2642). IEEE.

[5] Mhmmadalewi (2023) *Binary classifications: CNN: +96%ACC*, *Kaggle*. Available at: https://www.kaggle.com/code/mhmmadalewi/binary-classifications-cnn-96-acc/input (Accessed: 05 November 2023).