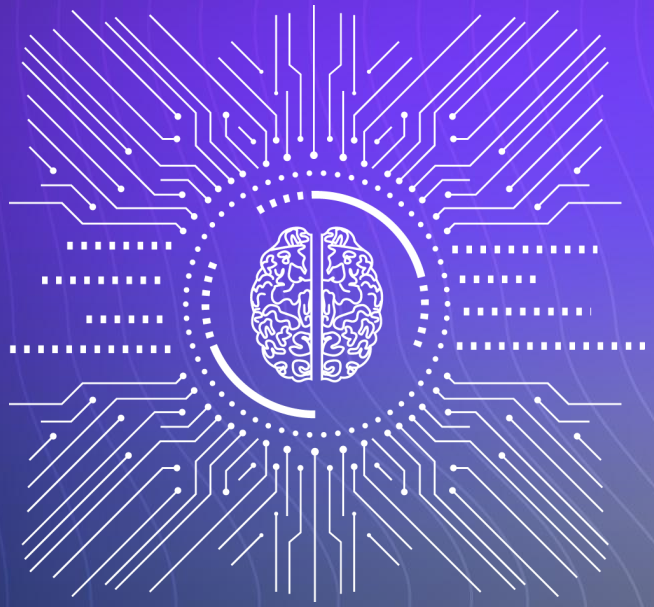


Transfer Learning
Group 14

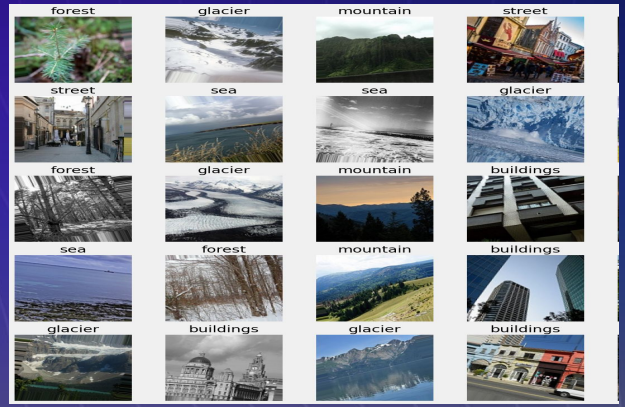
Moronfoluwa Akintola (22048063)
Kinza Hassan (22061674)
Tinubu Gabriel Oluwasogo (22065666)
Roseline Abosede Alao (22051527)
Oladeji Blessing (22061475)
Mansoor Imtiaz (22046808)

Google Colab Data Source Github

Introduction to the dataset

The Intel image classification dataset is a collection of images used for training machine learning models to recognize and classify different scenes. The dataset contains about 25,000 images in a 150 x 150 shape and it is categorized into six classes: Buildings, Forest, Glacier, Mountain, Sea, and Street. Each image in the dataset is labeled with its corresponding class, making it a valuable resource for machine learning models. The diversity and quality of the images in the dataset make it suitable for developing robust models capable of accurately identifying various natural scenes.

The images are split into training, testing, and prediction sets for building and evaluating models. For our analysis, the dataset was split into 14,034 images for training and 3,000 images for testing.



Overview of Transfer learning

Transfer learning is a machine learning technique where a model developed for a specific task is reused as the starting point for a model on a second task. This approach is highly beneficial when dealing with limited data for a new task, as it leverages the knowledge gained from a large previously solved problem to achieve better performance with less data. It can be traced back to educational psychology, where psychologist C. H. Judd introduced the generalization theory of transfer. According to this theory, the ability to transfer learning from one situation to another stems from the generalization of experiences. Individuals can apply skills and knowledge in one context to different situations as long as they can generalize their experiences [1].

Transfer learning not only accelerates the development process but also enhances the generalization capabilities of models to new, unseen data. By enabling the reuse of existing models, transfer learning fosters more efficient and effective machine learning workflows across a wide range of applications. It is very popular in deep learning applications such as image and speech recognition, where pre-trained models on extensive datasets can significantly reduce training time and computational resources.

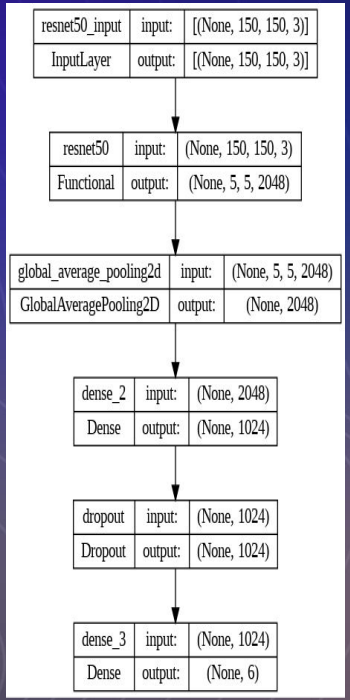
Importance of Transfer learning

It is useful in scenarios where the dataset for the second task is too small to train a full-scale model from scratch effectively. By leveraging the knowledge gained from the first task, transfer learning allows for significant improvements in learning efficiency and performance on the second task, often with less computational resources and time. The approach is popular in deep learning, especially in fields like computer vision and natural language processing where pre-trained models on large datasets are available.

Transfer learning is important because it enables the application of machine learning models to a broader range of problems, including those with limited data. It also facilitates rapid prototyping and development by allowing researchers and software engineers to build upon existing, proven models. It also eliminates the need for extensive labeled datasets for each new model by allowing the use of pre-existing, pre-trained models, especially beneficial when dealing with unlabeled data. This method makes the development process of machine learning more streamlined and improves the speed of deployment, as it facilitates the recycling of acquired knowledge across different models.

Pre-trained model and its original purpose

The Residual Network (ResNet) model, is a type of Convolutional Neural Network (CNN) that was introduced by researchers at Microsoft Research. It was designed to solve the problem of vanishing gradients in deep neural networks by introducing residual blocks, which allow for the training of much deeper networks by using skip connections or shortcuts to jump over some layers. The original purpose of ResNet was to significantly improve the performance of image classification tasks on large-scale datasets like ImageNet, where it achieved remarkable success, winning the 2015 ImageNet Large Scale Visual Recognition Challenge (ILSVRC). These skip connections help to preserve the gradient flow through the network, making it possible to train networks with over a hundred layers effectively. ResNet models come in various depths including ResNet-50, ResNet-101 and ResNet-152, indicating the number of layers in the network.

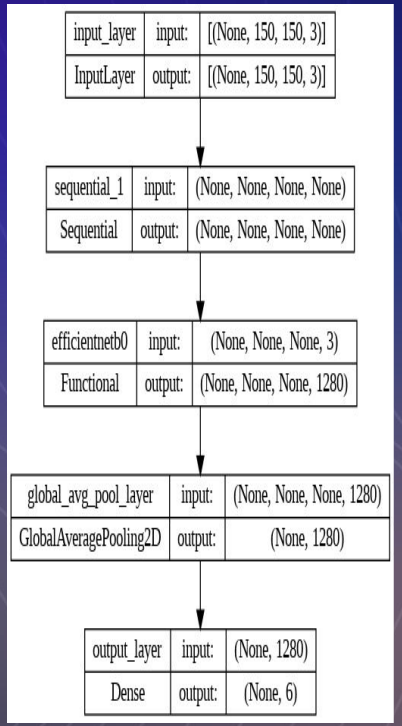


RestNet model architecture

Pre-trained model and its original purpose

The EfficientNet model is a family of Convolutional Neural Networks (CNNs) designed to achieve state-of-the-art accuracy on image classification tasks while being highly efficient in terms of computational resources. It was developed by researchers at Google, the original purpose of EfficientNet was to provide a scalable and efficient solution for deep learning models that could perform well across a wide range of computing devices and constraints.
It was trained on the ImageNet dataset, making it highly versatile for various image classification tasks.

EfficientNet strikes a balance between network width, resolution, and depth through the composite scaling coefficient, ensuring that the model has superior performance as the network scales up in complexity [2]. This approach allows EfficientNet to achieve much higher accuracy with significantly fewer parameters and computational power compared to previous models.

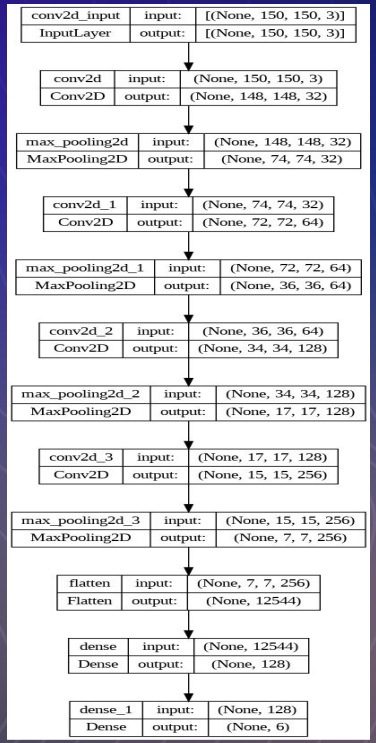


EfficientNet model architecture

Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) is a deep learning algorithm that takes an input image, assigns importance (learnable weights and biases) to various objects in the image, and can differentiate one from the other. The architecture of a CNN is designed to mimic the way the human brain processes visual information, using a series of layers that filter and pool the input image to create a hierarchy of features.

We implemented a CNN from scratch to benchmark against the pre-trained models. Our CNN architecture consists of an input layer that takes an image of size 150x150 with 3 color channels (RGB). It then passes through multiple convolutional layers (Conv2D), where filters are applied to detect spatial hierarchies of features, and max pooling layers (MaxPooling2D), which reduce the spatial dimensions of the feature maps. After several convolutional and pooling layers, the feature maps are flattened into a single vector and passed through fully connected layers (Dense), culminating in an output layer that makes the final classification which is 6 classes in our case.



CNN model architecture

ResNet Implementation

Breakdown of the steps taken from setup to fine-tuning the ResNet model.

1. Loading a Pre-trained Model:
   - The ResNet50 model, pre-trained on the ImageNet dataset, is loaded without its top layer because the top layer is specific to the original classification task and our new task has a different number of classes.
2. Freezing the Base Model Layers:
   - The weights of the pre-trained model are frozen (resnet_base_model.trainable = False). This means that these weights will not be updated during the training of the new top layers. This will prevent the loss of the generic features learned by the model on the initial ImageNet dataset.
3. Adding Custom Layers for the New Task:
   New layers are added on top of the pre-trained model to adapt it to the new classification task. These layers typically include:
   - A GlobalAveragePooling2D layer to reduce the dimensionality of the feature maps.
   - A Dense layer with a ReLU activation function to learn non-linear combinations of the features.
   - A Dropout layer to reduce overfitting by randomly setting input units to 0 during training.
   - A final Dense layer with a softmax activation function, where the number of units equals the number of classes in the new task.
4. Compiling the Model:
   - The model is then compiled with an Adam optimizer, categorical_crossentropy loss function, and a metric to monitor the accuracy.
5. Training with Early Stopping Callbacks:
   - The model is trained on the new dataset with early stopping callbacks monitoring validation loss and accuracy. These callbacks halt the training process if there's no improvement in the monitored metric for a specified number of epochs. This approach will help us prevent overfitting and save computational resources.

EfficientNetB0 Implementation

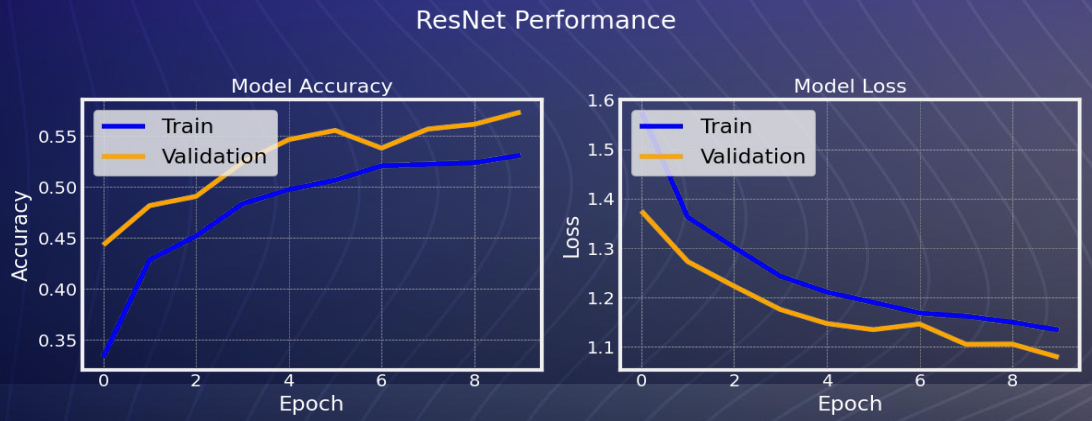
Breakdown of the steps taken from setup to fine-tuning the EfficientNetB0 model.

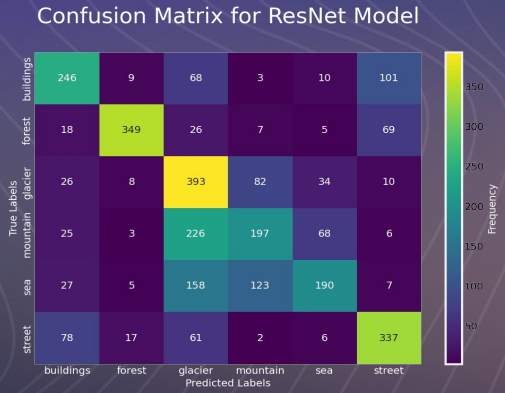
1. **Base Model Loading and Freezing:**
   - The EfficientNetB0 model is loaded with its pre-trained weights but without the top layer, using tf.keras.applications.EfficientNetB0(include_top = False). This is because the top layer is specific to the original training task, which is different from the new task.
   - The trainable attribute of the base model is set to False to freeze its weights. This means that during the initial training phase, the weights of the pre-trained model will not be updated. This step is crucial to avoid losing the pre-trained features during the early stages of training on the new dataset.
2. **Added Custom Top Layers for the New Task**
   - After preprocessing the inputs with a data augmentation layer, the base model processes the input images. Since the base model is frozen, its weights remain unchanged during this phase.
   - A Global Average Pooling 2D layer follows the base model's output. This layer reduces the dimensionality of the feature maps and prepares them for classification.
   - The final layer is a Dense layer with a softmax activation function. The number of units in this layer matches the number of classes in the new task. This layer is trainable and will be adjusted to classify images according to the new dataset.
3. **Initial Training**
   - The model is compiled with a categorical crossentropy loss function and Adam optimizer. At this stage, only the weights of the top layers are trainable.
   - The model is trained on the new dataset for 10 epochs, using callbacks for TensorBoard and model checkpointing. During this phase, only the top layers adjust their weights to the new data.
4. **Unfreezing and Fine-Tuning**
   - The unfreezing is done by setting trainable = True for the base model.
   - The model is then compiled again, often with a lower learning rate to fine-tune the pre-trained weights, and undergoes further training, where both the newly added top layers and the unfrozen layers of the base model are adjusted to better fit the new task.

ResNet Evaluation

The value 246 in the top left corner of the confusion matrix indicates that the model predicted 246 images that were buildings to be buildings. Looking at the diagonal of the confusion matrix, which runs from the top left to bottom right, we can see the number of correct predictions for each category. For example, the value 393 on the diagonal indicates that the model correctly classified 393 glacier images. Higher values on the diagonal are better, as it indicates that the model is accurately classifying those categories.

The values off the diagonal represent misclassified images. For example, the value 68 in the top row, second column indicates that the model incorrectly classified 68 forest images as buildings. Similarly, the value 101 in the bottom left corner indicates that the model incorrectly classified 101 street images as buildings. By analyzing the confusion matrix, we can identify which categories the model is most likely to confuse. In this case, the model seems to be most confused between buildings and streets (101), as well as between mountains and glaciers (82). The model also frequently confuses forests with buildings (68) and streets (158).



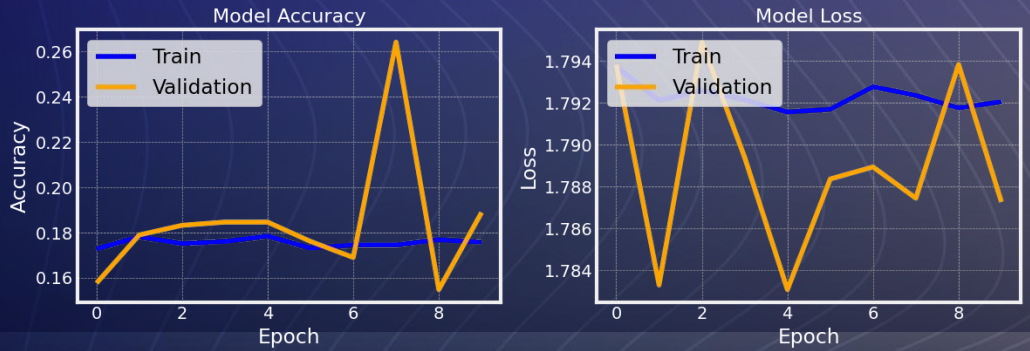


EfficientNet Evaluation

From the charts below, the validation accuracy is at 0.26, which means the model is classifying 26% of the images correctly. The validation loss is slightly lower than the training loss, at 1.792 compared to 1.794. This suggests that the model may be slightly overfitting the training data. We can see that the model is performing well for some labels (e.g., glacier) and not as well for others (e.g., buildings). The EfficientNet model in this graph appears to have reached a point where it is not improving much further on this dataset.

Classification reports are a way of evaluating the performance of a model on a classification task. They provide different metrics including precision, recall, F1-score, and support. The precision for class 1 is 0.529, which means that 52.9% of the time that the model predicted class 1, it was correct. The recall for class 1 is 0.814, which means that the model identified 81.4% of the actual class 1 cases. The overall accuracy of the model is 0.189, which means that 18.9% of the time, the model made a correct prediction. This is a relatively low accuracy, and it suggests that the model is not performing very well.

EfficientNet Performance



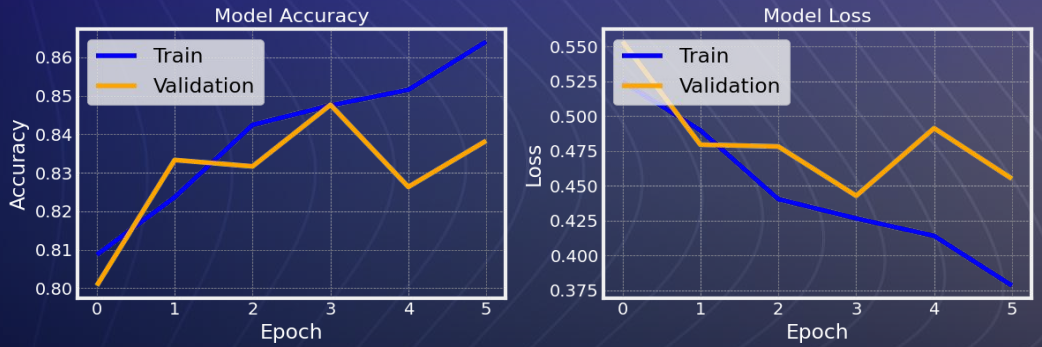
	precision	recall	f1-score	support
0	0.027985	0.034325	0.030832	437.000000
1	0.529492	0.814346	0.641729	474.000000
2	0.014925	0.003617	0.005822	553.000000
3	0.091220	0.152381	0.114123	525.000000
4	0.130548	0.098039	0.111982	510.000000
5	0.102639	0.069860	0.083135	501.000000
accuracy	0.189333	0.189333	0.189333	0.189333
macro avg	0.149468	0.195428	0.164604	3000.000000
weighted avg	0.145785	0.189333	0.159850	3000.000000

CNN Evaluation

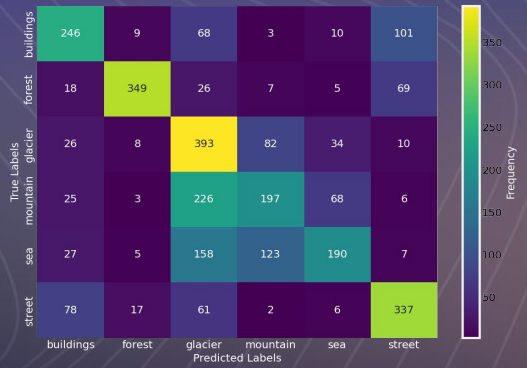
From the chart below, we would notice that the model's accuracy appears to be increasing over time, while the loss is decreasing. It indicates that the model is learning to correctly classify images. On the training set, the accuracy is around 86% after 5 epochs. This means that the model is correctly classifying 86% of the training images. The validation accuracy is slightly lower, at around 85%. This is because the validation set is a separate set of images that the model has never seen before. The validation accuracy is a more realistic measure of how well the model will perform on new data.

The confusion matrix shows the performance of the model, at the bottom right corner of the matrix, we can see that the model predicted 337 street images correctly, and misclassified 78 images as forests, 17 images as buildings, 61 images as glaciers, 2 images as mountains, and 6 images as sea. The model performed well on street images with 337 correct classifications and relatively low misclassifications. On the other hand, the model seems to have struggled more with forest and glacier images, misclassifying a significant number of them as other classes.

CNN Performance

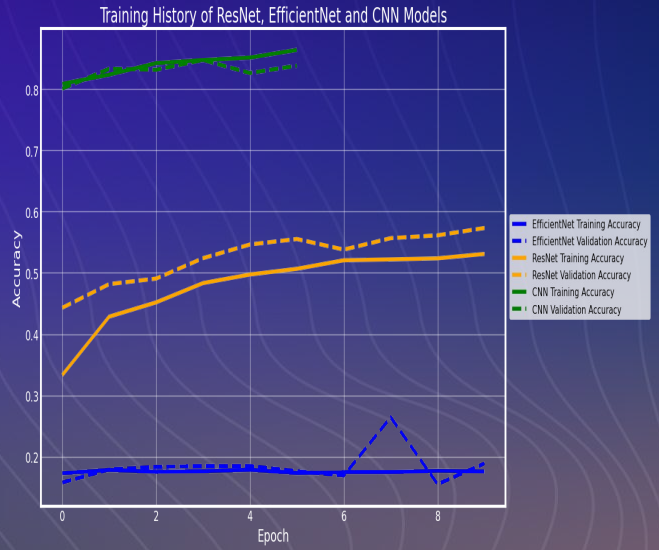


Confusion Matrix for Custom CNN Model



Comparison of results: Transfer learning vs from scratch

The image shows a graph comparing the training history of three image classification models: EfficientNet, ResNet and a custom CNN built from scratch. The x-axis represents the epoch, which is one pass through the entire training dataset. The y-axis represents accuracy. We would notice that as the number of epochs increases, the training accuracy increases and the validation accuracy increases. This indicates that the model is learning the patterns in the training data.



In the graph, the custom CNN appears to have the best performance, reaching a training accuracy of close to 0.8 around 2.5 epochs. ResNet also seems to achieve a similar validation accuracy but takes more epochs to train. The EfficientNet model in the graph appears to have the lowest accuracy of the three models. It is important to note that the optimal number of epochs can vary depending on the dataset and model complexity. Overfitting can occur if the model is trained for too many epochs.

Limitations and potential areas of improvement.

1. **Data Similarity:** The effectiveness of fine-tuned EfficientNetB0 and ResNet models depends on how closely the new task's data resembles the data from the original ImageNet dataset.
2. **Model Complexity for Task:** While EfficientNetB0 and ResNet are powerful for image processing tasks, their suitability varies based on the complexity of the task and dataset size, and alternative architectures might offer improved results.
3. **Training Data Quantity and Quality**: The performance of the model depends on both the quantity and the quality of the training data. An increase in dataset size or data quality through meticulous cleaning and preprocessing can boost the model's precision.
4. **Data Augmentation Scope**: The current data augmentation techniques may not cover all possible variations in real-world images. Implementing more augmentation to include diverse transformations would improve model robustness.
5. **Fine-Tuning Strategy**: Optimizing the fine-tuning approach by adjusting the number of layers unfrozen in the base model or modifying the learning rate during fine-tuning would effectively tailor the pre-trained weights to the specific requirements of the new task.

References

1. Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H. and He, Q., 2020. A comprehensive survey on transfer learning. Proceedings of the IEEE, 109(1), pp.43-76.

2. Hu, J., Shen, L. and Sun, G., 2018. Squeeze-and-excitation networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7132-7141).