Intel Image Classification

Aviya Oren 322273301
Maya Hayat 322515669

Abstract

This paper explores the task of classifying natural scene images using both logistic regression and convolutional neural networks (CNNs). We compare the performance of these two approaches on a dataset of 14,034 RGB images (150x150x3) categorized as buildings, forests, glaciers, mountains, seas, and streets.

We specifically focus on demonstrating the power of CNNs for image classification. We will build and analyze a CNN model, compare its performance to a logistic regression model, and discuss potential areas for improvement.

Compared to the logistic regression model, which achieved an accuracy of 46%, the CNN model significantly improved performance, reaching an accuracy of an impressive 88.2%.

This paper delves into the code, analyzes the model's components, and explores both its achieved performance and potential areas for future advancement.

Introduction

Scene classification has emerged as a crucial technology with broad applications across various domains. From enabling autonomous vehicles to navigate diverse environments to organizing our digital memories based on their content, scene classification offers a powerful tool for unlocking the meaning and context within images.
This project aims to classify images into six categories, the data is found on Kaggle, at https://www.kaggle.com/datasets/puneet6060/intel-image-classification.

The problem was originally approached using the well known softmax regression for multi-classes. This model trains a separate linear model for each class. Each model has its own weight vector and bias, and the linear outputs are passed through the softmax function to obtain probabilities for each class. The softmax function normalizes the outputs to ensure they sum to 1 and represent valid probabilities.

Next, we introduced a more complex and suitable model named CNN, the model offers a greater range of abilities, especially in its capacity to capture complex spatial and contextual features within the images. This allows it to differentiate between subtle details and patterns that might be missed by simpler models.
The initial CNN model was improved using the VGG pretrained model which indeed resulted in the highest accuracy.

This paper explores the importance of data cleaning and transfer learning for the scene classification problem. It compares different models and highlights the benefits of utilizing pre-trained models to achieve better results.

Related work and background research

Loss Function

In machine learning, the loss function acts as a crucial guide for model improvement. It quantifies the discrepancy between a model's predictions and the true, desired outputs for any given input example. Essentially, it measures how inaccurate the model's predictions are. This numerical value, the loss, is then used by optimization algorithms like gradient descent to adjust the model's internal parameters. The goal is to minimize the loss over time, leading to progressively better and more accurate predictions. In simpler terms, the lower the loss, the closer the model's guesses are to the actual answers. This iterative process of prediction, loss calculation, and parameter adjustment is fundamental to training effective machine learning models.

Sparse categorical cross entropy

The loss function used as the model contains numerous classes and this function specifically for multi-class classification problems. The function compares the predicted probability distribution with the one-hot encoded true label, where One-hot encoding is a technique where each class is represented by a vector of zeros, except for the actual class index which is set to one. Next, it calculates a numerical value representing the mismatch between the classes.

Adam

It's an optimization algorithm that operates on the gradients calculated based on the loss function. This helps avoid getting stuck in local minima and navigate more effectively towards the optimal solution.
As learned, in machine learning the goal is often to train a model that minimizes a loss function. Reaching the minimum point of the loss function signifies the optimal solution, where the model performs best on unseen data.
Adam Optimizer doesn't directly modify the loss function, but it utilizes information derived from gradients to minimize loss function but avoid stucking in local minima.
It uses gradient information, also it dynamically adjusts learning rates for each parameter.

Softmax Function

In multi-class classification tasks, the softmax function plays a crucial role in transforming raw scores, often representing the model's outputs, into interpretable probabilities. It takes a vector of real numbers as input and converts them into a probability distribution across the possible categories. These probabilities range from 0 to 1, ensuring they sum up to 1, and represent the likelihood of each class belonging to the input data. The final classification (as we can see in the accuracy calculation) is the maximum probability.

logistic regression

We will concentrate on multi-class logistic regression (also known as softmax regression) because simple logistic regression is a statistical method used for binary classification problems.
Instead of a single linear model, multi-class logistic regression trains a separate linear model for each class (let's say we have K classes).
Each model has its own weight vector and bias:

```
z_k = w_k^T * X + b_k  (for each class k = 1, 2, ..., K)
```

The linear outputs (z_k) are passed through the **softmax function** to obtain probabilities for each class. The softmax function normalizes the outputs to ensure they sum to 1 and represent valid probabilities.
P(y = k | X): The probability of X belonging to class k.

VGG

The VGG (Visual Geometry Group) architecture is a deep convolutional neural network (CNN) renowned for its simplicity and efficiency. This architecture utilizes primarily small 3x3 filters (kernels) throughout its layers, contributing to its computational efficiency and ability to learn complex features.
The main components of VGG architectures (we used VGG16) are:

- Convolutional layers: These layers use the small 3x3 filters to extract features from the input image. Multiple convolutional layers are stacked to create increasingly complex feature representations.
- Max Pooling Layers: These layers are used to downsample the spatial dimensions of feature maps. This helps reduce overfitting (making the model too specific to training data) and makes the model more amenable to computational approach.
- Fully Connected Layers: The final layers of the VGG network are fully connected. These layers take the features extracted by the convolutional and pooling layers and use them to make the final classification prediction.
  This creates a fully interconnected network of neurons, allowing them to combine information from the entire previous layer, not just localized regions.

VGG uses three fully connected layers as the final stages of its architecture.
These layers receive the flattened output from the previous convolutional and pooling layers, which transforms the multidimensional feature maps into a single, long vector.
Each fully connected layer performs a matrix multiplication between the input vector and its own weight matrix, followed by an activation function (like ReLU).
The final fully connected layer typically has one neuron for each class the model is trying to distinguish.
The activation function (often softmax) applied to the output of this final layer produces a probability distribution across all classes, indicating the model's confidence level for each class based on the learned features.

VGG models achieved outstanding results in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2014. This demonstrated the effectiveness of deep CNN architectures for image classification tasks.

Therefore, we determined the VGG model to be ideally suited for our mission of image classification.

Fine tune models

This technique allows us to capitalize on the knowledge gained by pre-trained models and adapt them to specific tasks.
Start with a pre-trained model: This model has been trained on a massive dataset for a broader task (e.g., image recognition). For instance CNN could be one of those pre-trained models.
Freeze initial layers: The initial layers of the pre-trained model, which contain foundational knowledge like recognizing edges and shapes, are often frozen (parameters remain unchanged). This preserves their valuable general knowledge.
Fine-tune final layers: The final layers, responsible for making specific predictions, are fine-tuned. This involves updating their parameters using a smaller dataset specific to your desired task (e.g., classifying view images). This allows them to learn the nuances of our specific problem.
The success of fine-tuning heavily relies on the quality and size of the task-specific data used for training.
There is an Overfitting risk: we need to avoid the model overfitting to the specific dataset.Fine-tuning can limit the model's ability to generalize to entirely new categories not included in the fine-tuning dataset.

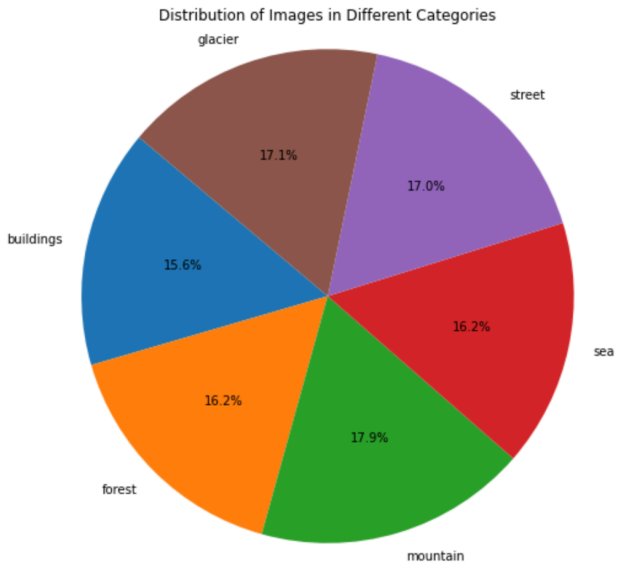
Project description and experiments

Preprocessing

We began the project by uploading the image data. Since the dataset consisted of over 15 thousand colorful images with a size of 150x150 pixels, we had to upload them in batches of 128 to manage computational efficiency. We maintained the original image size throughout the process.

Initially, the dataset included six categories: buildings, forests, glaciers, mountains, seas, and streets. We normalized all images so their values are between 0 and 1 instead of 0 to 255 to ensure consistency in pixel values across the dataset.
The decision to not make too many changes to the data during the preprocessing came from the fact that we had a massive amount of data therefore the data was kept as is throughout the process.

Before starting to create the models and train them, checking whether the data division is indeed balanced is a very important part of the project as we would like to avoid classifying images wrong for the sole reason that the data is imbalanced. The graph below shows how our data is splitted into fairly even categories.



Project description

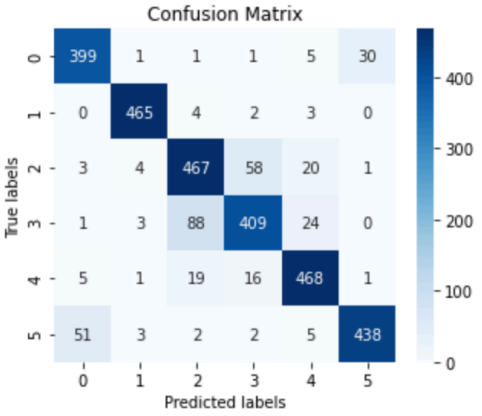
This project aims to classify view images into six categories: sea, mountains, forest, glaciers, buildings, and streets. This section will briefly discuss the best model and its results.

The Fine-Tuned VGG16 model created for Image Classification with six categories succeeded in getting an accuracy of 88.2% which is a great improvement both from the initial CNN model created but especially from the Logistic regression mode.

The model is built using the VGG16 model as follows.
First there is the VGG16 model which is pre-trained on the ImageNet dataset served as the base model. Its layers are frozen to prevent retraining as the training time already surpasses 6 hours (this of course would differ between computers, however with our resources we've made the decision of freezing those layers). In this model, unlike in the previous ones, we loaded our data as 224 by 224 which is the original size of the images in the ImageNet,the initial thought is that reusing the same input shape allows the pre-learned features to be more effectively transferred to the current classification.

Head Model: A new head model is created, consisting of a flattening layer which transforms the output of the base model into a 1D vector, next we created a fully connected layer with 128 neurons, using the ReLU activation function and last but not least the output layer which consists of 6 neurons and the softmax activation layer so it can easily classify the images into 6 classes.

After having received the results, the confusion matrix showed that the model mainly confuses the mountain and the glacier, therefore a deeper look was taken into the glacier and mountain images which raised a few concerns since there's a great overlap between the 2 and also since the glacier imageset is dirty, in the sense of it contains unrelated images as shown in Appendix A.



Another trial was created in which the glacier data was completely removed and the same VGG16 model was applied, with the same added layer, however, this time we kept the original image size.

The model managed to achieve an accuracy of 92.4%.

Previous attempts

The research has begun using the “Dummy-model” which randomly assigns each image in the test set to one of the specified categories.This model succeeded in 19.25% of the time which made sense as it assigns labels randomly and since we had 6 classes, the model would likely to be wrong.
First, we tried the Logistic regression model, since we are trying to classify into 6 classes, the model used was the One-vs-Rest. One must notice that our data is fairly difficult in the sense of it having many colours overlaps between classes as well as the fact that the images do not necessarily have the images in each class, meaning no consistent patterns.

After running the Logistic regression model, the current Multi-Layer Perceptron (MLP) model utilizes a simple architecture with a Flatten layer to convert the input images into a 1D vector, followed by a Dense layer with 128 neurons and ReLU activation for feature extraction, and a final Dense layer with 6 neurons and softmax activation for predicting the 6 classes. However, this baseline model currently achieves an accuracy of only 50%, indicating that further exploration and improvements are needed.

The next part of the project was introducing the CNN model. The first model used employed a sequential architecture with three convolutional layers, each followed by a max pooling layer. The convolutional layers utilized 3x3 kernels and ReLU activation functions. This combination is used to extract features from different levels of abstraction. After the convolutional layers, the model employed a flattening layer to convert the feature maps into a 1D vector. Finally, two fully-connected layers were added: the first with 128 neurons and ReLU activation, and the second with six neurons (as we’re classifying into 6 classes) and Softmax activation for multi-class classification. The first model resulted in a great improvement from the first logistic regression model and returned a 79.73% accuracy.

Since there was much room for improvement, we decided to use the famous VGG model. We first started experimenting with the model that was loaded with the 'imagenet' weights, transferring the pre-learned feature representations. The option include_top=False was used to exclude the final classification layers of VGG16, which were specific to the ImageNet categories, therefore allowing us to adapt the model to the current classification. The model was added another dense layer with 128 neurons and ReLU activation, this layer takes the flattened output from the pre-trained model and projects it into a new space with 128 features.The model is trained on the pre chosen train set and validation set. This experiment resulted in 87.4% accuracy.

In the third and last trial, the model was kept the same as it was resulting in a great accuracy (considering the data), however we did change the size of the input images to be the same size as the images in the ImageNet dataset which is 224x224 as the initial thought was that the pre trained model would have easier time finding the important features in the new images as it was trained on images the exact same size.

After viewing the model’s main confusion, it was decided to run our models on the data excluding the glacier images.
The first model managed to reach 84.6% accuracy and when using the VGG16 just as was previously used, keeping the input size at 150 by 150, the model managed to reach up to

92.4% accuracy which is a great improvement especially when looking back at the logistic regression model we initially started with.

Conclusion

In conclusion, this project has proved to us the importance and great strength of the CNN model, especially compared to the more basic models precisely, the Logistic regression model. The project continued and managed to explore ways to improve the first created model by leveraging a pre-trained VGG model. We decided to use a pretrained model as features, like edges, textures, and basic shapes, are often generic and can be valuable for various classification tasks, including yours. By utilizing pre-learned features, the model can potentially achieve better performance compared to starting from scratch. As expected, the new model was able to predict over 88% of the images correctly, compared to the initial CNN model which predicted 79.7%.

There were numerous challenges encountered throughout the project, this section will highlight some and will discuss their solutions.

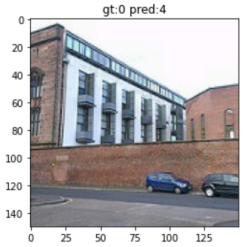
The first and main issue was encountered was the glacier's dataset, as shown in Appendix A, the glacier dataset consists of images that should fall under other categories and has many overlaps with the mountain dataset as there are many images of mountains covered in snow in both folder, therefore the model cannot know where those images belong. We've decided to run all our models on the same data, excluding the glacier images which did indeed increase our results to 92.4%.

The second limitation encountered was the size of the data. With nearly 15 thousand images, the model was unable to train on the entire dataset at once due to memory limitations. To address this, we implemented a batch processing approach, where the data was divided into smaller batches of 128 images. While this allowed training to proceed, it also necessitated reloading each batch to the RAM before training on it. This repeated loading process significantly increased the overall training time compared to training with the entire dataset in memory at once.

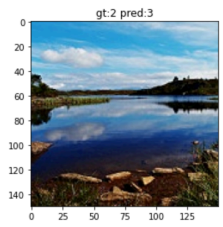
Overall, this project demonstrated the strength of the CNN model and provided us with a great view of the variety of available pretrained models that could be adjusted to fit new problems. Our finding demonstrated that the VGG16 model where the input data was of shape 224 by 224 achieved an accuracy of 88.2%.
This result could possibly be improved by looking further into more models that could be taken into account as we did with the VGG16, e.g VGG19 or ResNet.

Another important thing to mention is the fact that there are some overlaps between the classes, as presented below where images that actually represent streets are found in the building folder, therefore, the model wouldn't possibly guess them correctly.





Another confusion one must understand is between mountains and sea, let's look at the following images and try to understand the cause. Note how the images below contain mountains, however there are also lakes found in the images, therefore it wouldn't be a great surprise/ mistake that the model would classify them under sea.

After a careful look at the data, we are more confident that our model is significantly better than the 88.2% it managed to achieve. An improvement we must keep in mind for future projects is to spend enough time going through our data and cleaning it. This process is exhausting and time consuming, however this is possibly the most important component of the model creation as one cannot possibly expect our model to learn and predict well if the data it was given is wrong.

Full code can be found on https://github.com/MayaHayat/Intel_Classification

Appendices

1. Appendix A
   The first three images are taken from the glacier image set and the last image on the right was taken from the mountain dataset.

Bibliography

1. Intel Image Classification (CNN - Keras), VINCENZOOO

   https://www.kaggle.com/code/vincee/intel-image-classification-cnn-keras
2. https://www.tensorflow.org/api_docs/python/tf/keras/applications/vgg16/preprocess_input