Intel Image Multiclass Classification

Ricardo Pinto rjsp15@ua.pt, Inês Castro icbaptista@ua.pt
University of Aveiro, Portugal
Department of Electronics, Telecommunications and Informatics (DETI)
Course: Machine Learning Topics, Course Instructor: Prof. Pétia Georgieva
Workload per Student (50/50): Ricardo's contributions include state of art [2], [3] and [4], data preprocessing and exploration and 2 normal CNNs from scratch; Inês reviewed state of art [5], [6] and [7] and implemented 4 pre-trained CNNs that use transfer learning from ImageNet (we will only analyze two of them).

***Abstract*—Over the past decade, image classification, which can provide assistance to address complex tasks such as planetary exploration and unmanned driving, has become a hot topic. As a subproblem of image classification, scene image classification has received increasing attention. In this paper, we review the state of the art in this specific subproblem and use the Intel Image Classification dataset, which contains images of many natural scenes around the world, to build and optimize 4 different CNNs capable of accurately predicting which type of scenery is present in an image (sea, glacier, street, etc).**

***Index Terms*—machine learning, intel image classification, image scene classification, multiclass, classification, multiclass classification, prediction, CNN, convolutional neural networks**

I. Introduction

Image scene classification is becoming more and more popular and has become one of the challenges in Computer Vision, since it is useful in many areas, such as image organization and tagging, environmental monitoring, content recommendation, visual search, augmented reality and virtual reality, tourism and travel planning and data-driven decision making.

Scene classification is a complex task that is different from simple image classification where there exists only one or two objects in an image.

This paper has 2 main objectives. The first is to explore and review the state of the art involving the dataset; thus we review 6 different works and compare them. The second objective is to build, optimize and compare 4 different CNNs capable of accurately predicting which type of natural scenery is present in an image by applying various ML techniques; thus we have a multiclass classification problem.

Finally, we present the results and our conclusions.

II. State of the art

In [2], Vincent Liu presents a pratical, code-based approach to this problem/dataset, by building a CNN (Convolutional Neural Network) with the Keras library. They build a very simple model and fit it to the train set (14034 examples/images) with 20 epochs, achieving a 0.78 accuracy on the test set (3000 examples/images), which shows slight underfitting. Then, with a confusion matrix, they conclude the classifier has trouble with 2 kinds of scenery: streets and buildings. However, the classifier predicts forests extremely accurately. They then extract features with VGG16 (Visual Geometric Group) trained on ImageNet and train a one-layer Neural Network on those features, obtaining an accuracy of 0.87, a 0.1 increase over the simple CNN. They then ensemble 10 neural networks and obtain a model with 0.89 accuracy. Finally, they fine tune VGG ImageNet, also obtaining a test accuracy of 0.89%.

In [3], the authors build an Xception-based CNN with Transfer Learning. In neural networks, Xception is an improved version of the Inception architecture. Transfer learning means that, instead of training the CNN from scratch, it uses pre-trained models or weights from ImageNet. They train it on the Intel Image Classification dataset and conclude that it significantly outperforms other methods such as Inception-V3. Not only that, the Xception shows greater robustness and ability in generalization with less overfitting problems. The Inception CNN has a slightly higher accuracy (91.8%) than the one of XCeption CNN (91.2%) and also less loss, however the former shows more overfitting, making the Inception model questionable and unreliable.

In [4], the model RepConv is proposed, which is a CNN with very few parameters (160390). RepConv's accuracy on the Intel Image Scene dataset (94.58%) is very close to the other 4 models compared, while having hundreds of times less parameters, the lowest depth, and only 10% of the epochs of the compared models. The authors go even further, recategorizing the Intel Image Scene dataset into a binary classification problem, where the goal is to distinguish real scenes (buildings, streets) from natural scenes (sea, glacier, mountain, forest). In this binary classification, RepConv achieved a test accuracy of 92.70% which is not previously mentioned in any other publication.

In [5], ensemble methods were employed along with transfer learning, progressive image resizing, Mixup Augmentation, LR Tuning, and the use of Places365 dataset weights. Additionally, kNN with embeddings was explored as an enhancement to the ensemble strategy. By training multiple models independently and combining their predictions through ensemble techniques, such as voting or weighted voting, the system's accuracy was improved. The accuracy was further enhanced by removing potentially mislabeled or confusing images from the training dataset based on predictions indicating multiple classes or extremely low/high confidence. This approach resulted in a final test accuracy of 0.963.

In [6], Rahimzadeh et al. proposed Wise-SrNet, a novel architecture that addresses the challenge of preserving spatial resolution in feature maps for image classification. By replacing the Global Average Pooling (GAP) layer with the Wise-SrNet architecture, which utilizes depthwise convolutional operations, the authors achieved significant improvements in convergence speed and accuracy. Their experiments on various datasets and models demonstrated notable increases in accuracy for different image resolutions.

In [7], the authors proposed an improved DenseNet network, which is another CNN architecture, by improving the initial convolution block and activation function of DenseNet's network structure. Based on the improved DenseNet, the image features of the training set are extracted. The experimental results show that the accuracy of the improved DenseNet algorithm is improved by 8.89%. While this paper used the CIFAR-10 dataset and not the Intel Image Classification dataset, studies like this help us understand how different state-of-the-art CNN architectures can are able to tackle the task of multi class image classification.

III. Dataset and preprocessing

The Intel Image Classification dataset contains around 17000 images of size 150x150 of 6 diferent types of natural scenes around the world: buildings, forest, glacier, mountain sea and street. The train and test sets contain 14034 and 3000 images, respectively. This means that 18% of the data is for testing. We split the train data into 80% for training (11227 images) and 20% for validation (2807 images). All images are normalized by dividing them by 255 (maximum value).

The images were loaded in batches into RAM and fed to the CNNs during training, while the CNNs themselves were loaded and trained on the GPU (Nvidia GeForce GTX 1650) of Ricardo's personal computer.

Fig. 1: Example image from the dataset for each class.

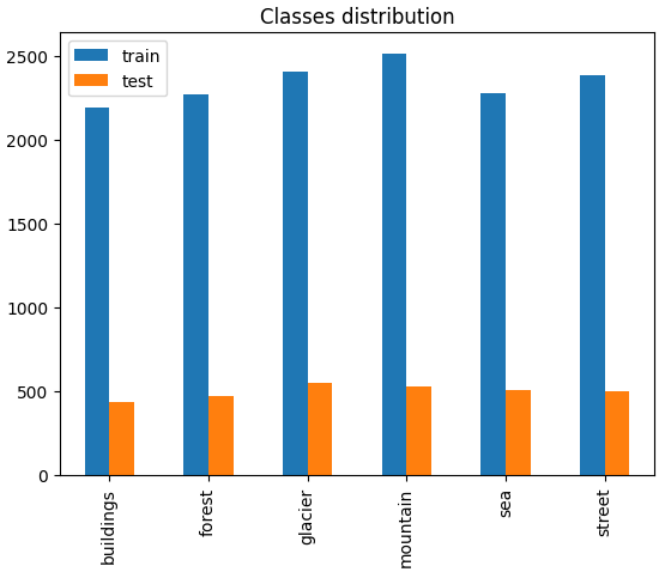

Fig. 2: Classes distribution

As we can see, both train and test sets are balanced, meaning accuracy is a good metric to evaluate our CNNs.

IV. CNNs

Using tensorflow and keras, we trained 4 different CNNs: 2 normal ones from scratch and 2 pre-trained (ResNet50 and VGG16). The pre-trained ones use transfer learning from ImageNet. We used the Adam optimizer for both, because of its popularity and adaptive learning rate. For each CNN, we ran a grid search with the initial learning rates 0.01, 0.001 and 0.0001 and the batch sizes 32 and 128.

Because of hardware limitations (since we used our personal computers), we trained for a max of 10 epochs. Each epoch took between 30 seconds and 1 minute. If the validation accuracy was too low (less than 0.2) or didn't improve for 5 consecutive epochs, we would stop the training. After training was finished, we extracted the model at the epoch with highest validation accuracy.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 148, 148, 32)      896

max_pooling2d_8 (MaxPooling  (None, 74, 74, 32)        0
2D)

conv2d_9 (Conv2D)            (None, 72, 72, 32)        9248

max_pooling2d_9 (MaxPooling  (None, 36, 36, 32)        0
2D)

flatten_4 (Flatten)          (None, 41472)             0

dense_8 (Dense)              (None, 64)                2654272

dense_9 (Dense)              (None, 6)                 390

=================================================================
Total params: 2,664,806
Trainable params: 2,664,806
Non-trainable params: 0
```

Fig. 3: First normal CNN from scratch

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_37 (Conv2D)           (None, 148, 148, 32)      896

max_pooling2d_37 (MaxPoolin  (None, 74, 74, 32)        0
g2D)

conv2d_38 (Conv2D)           (None, 74, 74, 64)        18496

max_pooling2d_38 (MaxPoolin  (None, 37, 37, 64)        0
g2D)

conv2d_39 (Conv2D)           (None, 37, 37, 64)        36928

max_pooling2d_39 (MaxPoolin  (None, 18, 18, 64)        0
g2D)

conv2d_40 (Conv2D)           (None, 18, 18, 128)       73856

max_pooling2d_40 (MaxPoolin  (None, 9, 9, 128)         0
g2D)

conv2d_41 (Conv2D)           (None, 9, 9, 128)         147584

max_pooling2d_41 (MaxPoolin  (None, 4, 4, 128)         0
g2D)

flatten_11 (Flatten)         (None, 2048)              0

dense_22 (Dense)             (None, 512)               1049088

dropout_5 (Dropout)          (None, 512)               0

dense_23 (Dense)             (None, 6)                 3078

=================================================================
Total params: 1,329,926
Trainable params: 1,329,926
Non-trainable params: 0
```

Fig. 4: Second normal CNN from scratch

As we can see, the first has around 2.7 million parameters, while the second one has half of that (around 1.3 million) but more layers. The second one also has regularization, with a 0.5 dropout layer.

	CNN 1	CNN 2
Train accuracy	93.08%	89.08%
Validation accuracy	78.74%	84.27%
Test accuracy	78.63%	85.13%
Best learning rate	0.001	0.001
Best batch size	32	32
Best epoch	5	8

TABLE I: Comparison of our normal CNNs from scratch

The table above suggests some overfitting since the train accuracy is higher than the validation accuracy, even for the second CNN which had regularization. However, CNN 2 has way less overfitting than CNN 1, because of the 0.5 dropout layer that contributed to regularization. Let's analyse CNN 2, which achieved the highest test accuracy.

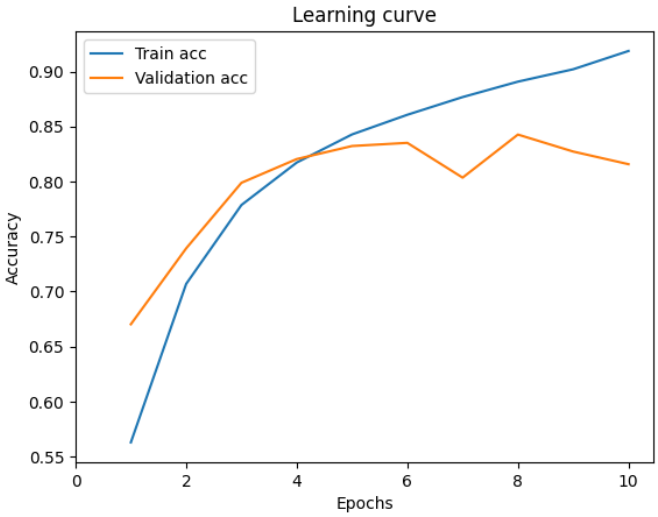

Fig. 5: Accuracy learning curve for CNN 2, hyperparameters optimized

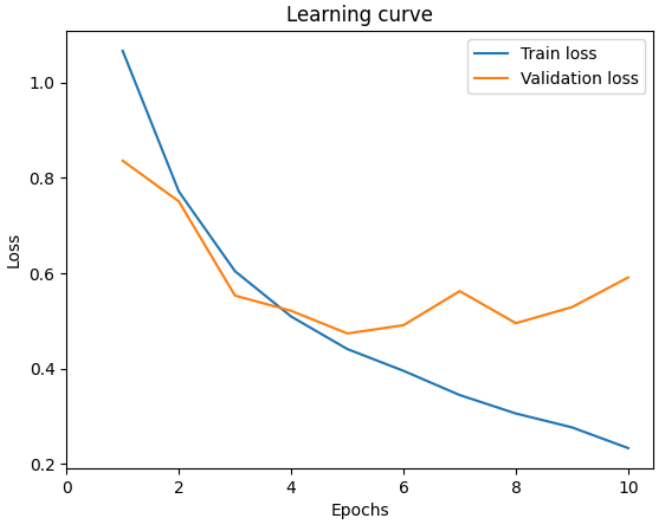

Fig. 6: Loss learning curve for CNN 2, hyperparameters optimized

In the learning curves above, we can see that overfitting starts to happen after epoch 5, since validation accuracy becomes lower than training accuracy. However, the highest validation accuracy was achieved at epoch 8, so that's the best epoch.

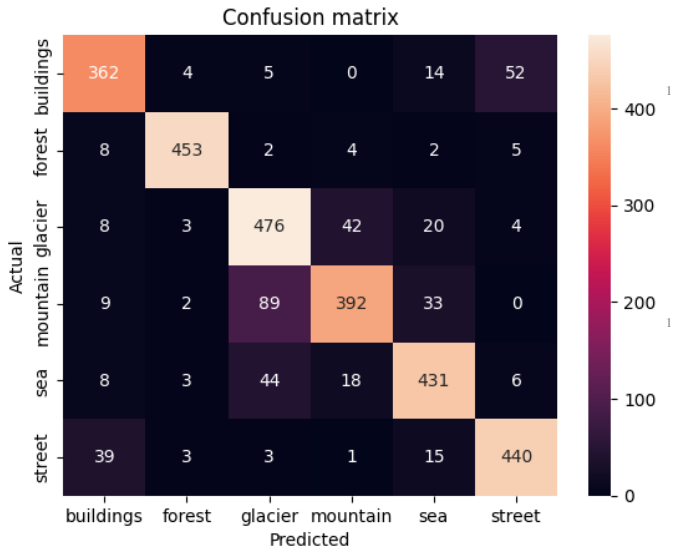

Fig. 7: Confusion matrix for CNN 2, hyperparameters optimized

In the confusion matrix above, we can see that this CNN is very good at correctly classifying forests, but not so good at distinguishing buildings from streets, glaciers from seas, and mountains from glaciers.

V. Transfer Learning CNNs

Some of the papers analyzed in the state-of-the-art, for example paper [5] use a technique called Transfer Learning. This technique involves using pre-trained models that are usually trained on massive datasets that are a standard benchmark in the computer vision field like ImageNet. The reusable weights of pre-trained models encode valuable knowledge about visual features and patterns, enabling their effective application to other smaller datasets.

Fig. 8: Transfer learning steps. [9]

As shown in Figure 8, the typical transfer learning methodology involves creating a pre-trained base model architecture like VGG-16 or Resnet-50, removing the last layers, using it as a feature extractor, and adding new task-specific layers that are fine-tuned on a smaller dataset. We will evaluate how two different architectures perform in the Intel Image Classification dataset.

A. VGG-16

The first pre-trained CNN architecture that we are going to analyze is VGG-16. The VGG-16 architecture is well-known for its simplicity and uniformity, consisting of multiple convolutional layers with small 3x3 filters and max pooling layers, followed by fully connected layers.

To implement the VGG-16 model, we used the Keras library and loaded the pre-trained weights trained on the ImageNet dataset. As shown in Listing 1, we employed the VGG16 model and excluded the top fully connected layers by setting includetop=False. We also adjusted the input shape to match our image dimensions of (150, 150, 3).

```
base_model = VGG16(weights='imagenet',
    include_top=False, input_shape=(150, 150, 3))
```

Listing 1: Loading the pre-trained VGG-16 model.

Then, we froze the layers of the base model by setting base-model.trainable = False to retain the pre-trained knowledge without updating the weights.

```
base_model.trainable = False
```

Listing 2: Freezing the base model's layers.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 4, 4, 512)         14714688

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)

 dense (Dense)               (None, 256)               131328

 dense_1 (Dense)             (None, 6)                 1542

=================================================================
Total params: 14,847,558
Trainable params: 132,870
Non-trainable params: 14,714,688
```

Fig. 9: CNN model built using a base pre-trained VGG-16 model

As shown in figure 9, on top of the base model, we have incorporated several layers for specific purposes. These include:

- Global Average Pooling Layer: calculates the average value of each feature map, creating a global representation of the input while reducing spatial dimensions.
- Dropout Layer: helps prevent overfitting by reducing interdependencies among neurons.
- Dense Layer with 256 Units and ReLU Activation: fully connected layer with ReLU activation function, enabling the model to capture complex nonlinear patterns.
- Final Dense Layer with 6 Units for Classification: last fully connected layer that generates the final output probabilities for each class.

The VGG-16 model has a total of 14,714,688 parameters, out of which 132,870 parameters are trainable and adjusted during the training process.

We fine-tuned the model by optimizing hyperparameters such as learning rate and batch size. After experimentation, we found that the Adam optimizer with a learning rate of 0.001, and a batch size of 32 yielded the best results.

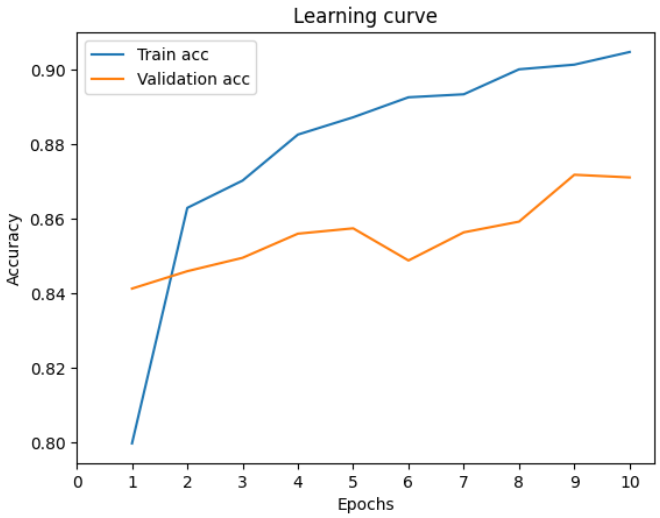

Fig. 10: Accuracy learning curve for VGG-16, hyperparameters optimized

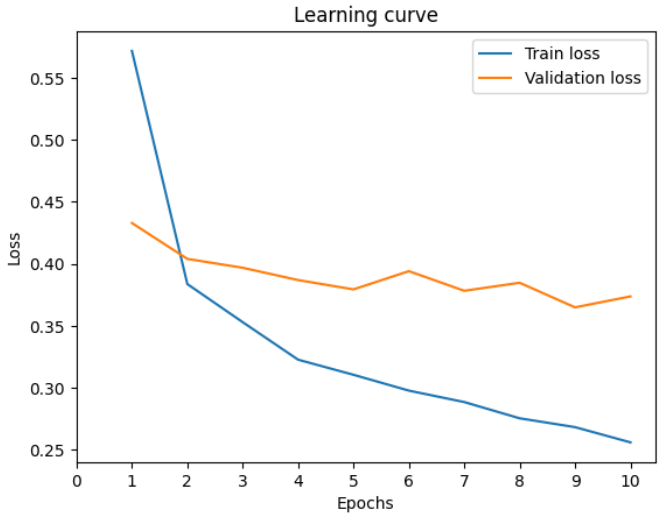

Fig. 11: Loss learning curve for VGG-16, hyperparameters optimized

In the learning curves above, we observe that the model starts to overfit after a certain number of epochs, indicating the need for further fine-tuning. Despite the observed overfitting, the model achieves a good validation accuracy of 87.18% and a high test set accuracy of 87.77%, indicating its ability to generalize well to unseen data.

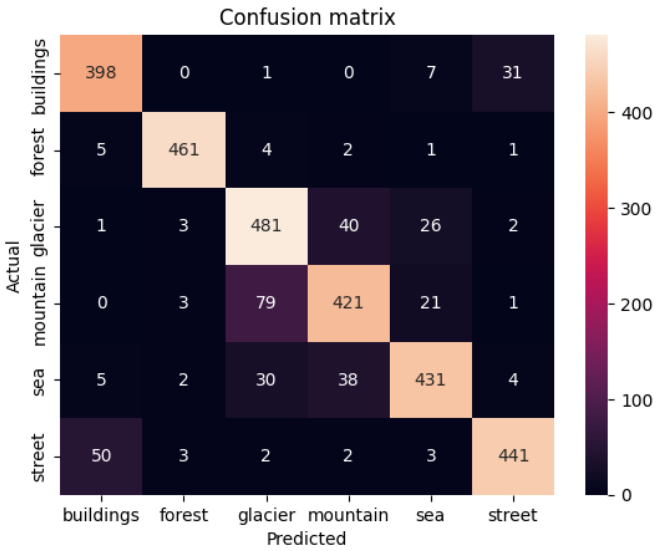

Fig. 12: Confusion matrix for VGG-16, hyperparameters optimized

Analyzing the confusion matrix above, we observe high accuracy in classifying forests (94.37%) and buildings (89.90%), while encountering challenges in accurately classifying glaciers (82.60%), mountains (84.30%), seas (84.63%), and streets (89.07%). The misclassifications are expected as for examples glaciers have similarities with the seas in terms of colour which can become confusing for the algorithm. Another example is the misclassification of glaciers as mountains which can be explained by the similarities between the outward shape of mountains and glaciers.

Overall, the VGG-16 model demonstrates promising performance, particularly in certain categories, while showing room for improvement in others.

B. ResNet-50 CNN

Another pre-trained CNN model we used to solve the task is ResNet-50. Compared to a normal CNN, ResNet-50 is a much deeper architecture with 50 layers. As seen in the state-of-the-art and as shown by the figure 13, this architecture incorporates skip connections and residual blocks to alleviate the vanishing gradient problem and capture more complex features, leading to improved higher accuracy in computer vision tasks. Weight updates follow the standard backpropagation with gradient descent.

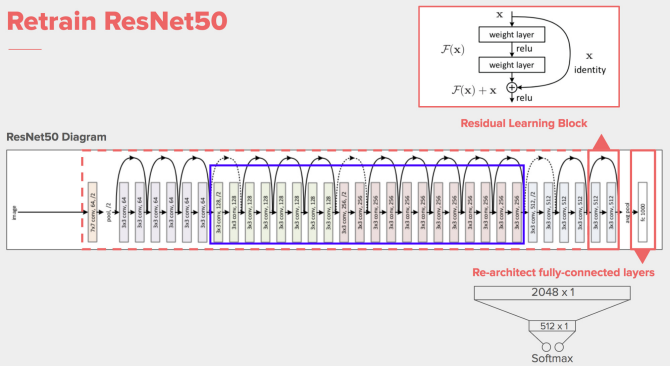

Fig. 13: Resnet-50 Model Architecture [8]

To implement this architecture, we used the TensorFlow and Keras libraries. The architecture utilized a base model, ResNet-50, pre-trained on the ImageNet dataset, with the last fully connected layer (includetop=False) excluded. The input shape was set to (150, 150, 3) to accommodate the image dimensions.

```
base_model = ResNet50(weights='imagenet',
    include_top=False, input_shape=(150, 150, 3))
```

Listing 3: Loading the pre-trained ResNet-50 model.

To acelerate training and prevent overfitting, the base model's layers were frozen (basemodel.trainable = False), retaining the pre-trained knowledge.

```
base_model.trainable = False
```

Listing 4: Freezing the base model's layers.

To prevent overfitting, a mew model was created on top of the base model. In the figure below, there is a representation of the layers of the top model.

```
Resnet-50 CNN Model
batch_size=32
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 5, 5, 2048)        23587712

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 6)                 12294

=================================================================
Total params: 23,600,006
Trainable params: 12,294
Non-trainable params: 23,587,712
```

Fig. 14: CNN model build using a base pre-trained ResNet-50 model.

As shown in figure 14, the base model used in this case has a total of 23,587,712 parameters. A global average pooling layer was applied, followed by a dropout layer. The model's final dense layer has 6 units corresponding to the classification classes, with 12,294 trainable parameters.

This combination of non-trainable and trainable parameters allows the model to leverage pre-trained knowledge while adapting to the specific task efficiently, making it less computationally demanding.

We fine-tuned the model by optimizing hyperparameters such as learning rate and batch size. After experimentation, we found that the Adam optimizer with a learning rate of 0.01 and a batch size of 128 yielded the best results. Interestingly, smaller learning rates obtained lower accuracies, with the best accuracies obtained with the highest learning rate tested (0.01).

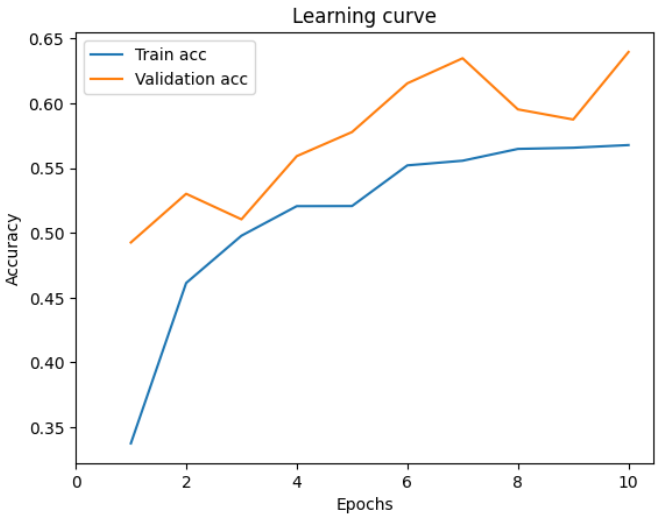

Fig. 15: Accuracy learning curve for ResNet-50, hyperparameters optimized

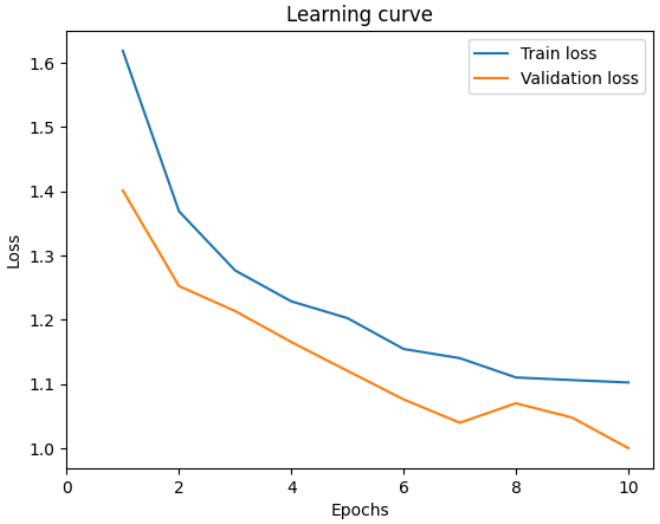

Fig. 16: Loss learning curve for ResNet-50, hyperparameters optimized

In the learning curves above, we observe that the model starts to overfit after a certain number of epochs, indicating the need for further fine-tuning.

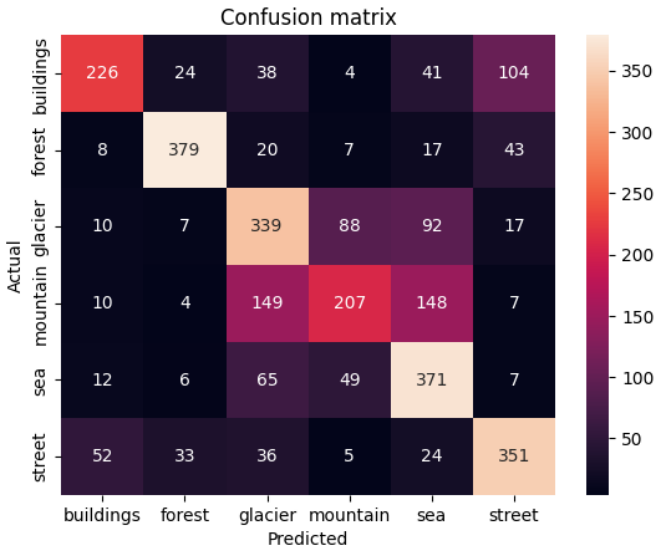

Fig. 17: Confusion matrix for ResNet-50, hyperparameters optimized

Analyzing the confusion matrix above, we can observe that the ResNet-50 model achieves high accuracy in classifying forests (94.37%) and buildings (89.90%). However, it encounters challenges in accurately classifying glaciers (82.60%), mountains (84.30%), seas (84.63%), and streets (89.07%). The misclassifications are similar to what was seen with the ResNet-50 model and can be attributed to similarities in color between glaciers and seas, as well as the resemblance in outward shape between mountains and glaciers.

Overall, the ResNet-50 model demonstrates promising performance, particularly in certain categories, while showing room for improvement in others.

VI. Results and Conclusions

	Test accuracy
[2] VGG16 / Ensemble NNs	89%
[3] Xception	91.2%
[4] RepConv	94.58%
ResNet50 [5]	96.3%
ResNet50 [6]	87.4%
DenseNet121 [6]	86.8%

TABLE II: Comparison of test accuracies of state of art

	CNN 1	CNN 2
Train accuracy	93.08%	89.08%
Validation accuracy	78.74%	84.27%
Test accuracy	78.63%	85.13%
Best learning rate	0.001	0.001
Best batch size	32	32
Best epoch	5	8

TABLE III: Comparison of our normal CNNs from scratch

	Resnet-50	VGG-16
Train accuracy	56.77%	90.13%
Validation accuracy	63.95%	87.18%
Test accuracy	62.43%	87.77%
Best learning rate	0.01	0.001
Best batch size	128	32
Best epoch	10	9

TABLE IV: Comparison of our pre-trained CNNs using transfer learning from ImageNet

A. Discussion

Out of the four, VGG-16 obtained the best train accuracy with 90.13%, the best test accuracy with 87.77% and the best validation accuracy with 87,18%. In the referenced papers, VGG-16 obtained a similar score but was outperformed by other architectures. Interestingly, the ResNet-50 model, which obtained the highest score in the referenced papers, got the lowest accuracies in our evaluation, which can be attributed to the model complexity and the low amount of epochs we used for training.

Analysing the confusion matrix of all 4 CNNs, we can see that all of them excel at classifying forests. However, they have some trouble distinguishing mountains and glaciers, which is expected as they have similar physical shapes. They also have some trouble distinguishing buildings and streets, which is expected since usually images of buildings also contain streets and vice versa. Finally, they have some trouble distinguishing seas and glaciers, since images of glaciers usually also show a sea and, not only that, the images of these 2 categories have a lot of blue in them (light blue, dark blue, etc) making it hard for the models to find identifying features. These conclusions are very similar to the conclusions in [2].

B. Challenges

The hardware limitations forced us to train the models for only 10 epochs. With more epochs, we would have achieved even better results. We addressed the challenge of limited hardware resouces and acelerated the training time by utilizing GPU instead of CPU, to leverage their parallel processing capabilities. We also loaded the images in batches into RAM to acelerate training which allowed us to keep the original image resolution and use the whole dataset.

C. Future iterations

In conclusion, our study analyzed state-of-the-art CNNs and architectures trained on the Intel Image Scene dataset, comparing methodologies and techniques to gain valuable insights into multi-class image classification and machine learning as a whole. Moving forward, other advanced strategies, referenced in the state-of-the-art, such as data augmentation, progressive resizing, and additional fine-tuning could be used to further enhance the performance and accuracy of our models.

VII. References

References

[1] Intel Image Classification Dataset https://www.kaggle.com/datasets/puneet6060/intel-image-classification

[2] Keras CNN on Intel Image Classification, by Vincent Liu https://www.kaggle.com/code/vincee/intel-image-classification-cnn-keras

[3] X. Wu, R. Liu, H. Yang and Z. Chen, "An Xception Based Convolutional Neural Network for Scene Image Classification with Transfer Learning," 2020 2nd International Conference on Information Technology and Computer Application (ITCA), Guangzhou, China, 2020, pp. 262-267, doi: 10.1109/ITCA52113.2020.00063.

[4] Soudy, Mohamed & Afify, Yasmine & Badr, Nagwa. (2022). RepConv: A novel architecture for image scene classification on Intel scenes dataset. International Journal of Intelligent Computing and Information Sciences. 1-11. 10.21608/ijicis.2022.118834.1163.

[5] A. Sayed, "1st Place Solution for Intel Scene Classification Challenge," in Medium, Dec. 2021. [Online]. Available: https://medium.com/@afzalsayed96/1st-place-solution-for-intel-scene-classification-challenge-c95cf941f8ed

[6] Rahimzadeh, M., Parvin, S., Safi, E., & Mohammadi, M. R. (2021). "Wise-SrNet: A Novel Architecture for Enhancing Image Classification by Learning Spatial Resolution of Feature Maps." In Proceedings of the IEEE International Conference on Image Processing (ICIP), 2021, 1-5. doi: 10.1109/ICIP42928.2021.9506420

[7] G. Wang, Z. Guo, X. Wan, and X. Zheng, "Study on Image Classification Algorithm Based on Improved DenseNet," J. Phys. Conf. Ser., vol. 1952, no. 2, p. 022011, Jun. 2021, doi: 10.1088/1742-6596/1952/2/022011.

[8] S. Mukherjee, "The Annotated ResNet-50 - Towards Data Science," Medium, Aug. 2022. [Online]. Available: https://towardsdatascience.com/the-annotated-resnet-50-a6c536034758

[9] D. Mwiti, "Transfer Learning Guide: A Practical Tutorial With Examples for Images and Text in Keras," Neptune, Apr. 2023. [Online]. Available: https://neptune.ai/blog/transfer-learning-guide-examples-for-images-and-text-in-keras

Appendix

The source code for our machine learning models and data preprocessing scripts is available on GitHub at the following repository: https://github.com/zzzzz151/TAA-Project2/tree/main.