Deep Learning and Reinforcement Learning

Course Project

Evgeny Zorin

9. November 2021

Intel Image Classification

Abstract

Identifying natural scenes from around the world is an interesting computer vision problem. How do humans recognize a forest as a forest or a mountain as a mountain? They are very good at classifying scenes based on semantic representation and object affinity, but they know very little about processing and encoding natural scene categories in the human brain. This project will demonstrate how the human brain classifies images using a Convolutional Neural Network.

Main Objective

The main objective of the project is to create and train a model based on a Convolutional Neural Network that is able to identify what type of natural scene an image can be classified as.

Project Workflow

1. Data Loading and Exploration
    - Data Summary
    - Data Visualization
2. Building Input Pipeline
    - Data Performance Configuration
    - Data Standardization
3. Modeling
    - Simple CNN Model (sCNN)
    - Augmented CNN Model (aCNN)
    - Complex CNN Model based on Pre-trained VGG16 Model (VGG16)
    - Model Comparison
    - Prediction
4. Key Findings and Insights
5. Next Steps

Data Loading and Exploration

Data Summary

The data was taken from Kaggle and it contains around 25k images of Natural Scenes around the world. All training and testing images with a size of 150x150 are classified into 6 categories:

- buildings = 0
- forest = 1
- glacier = 2
- mountains = 3
- sea = 4
- street = 5

The data consists of 3 separated datasets:

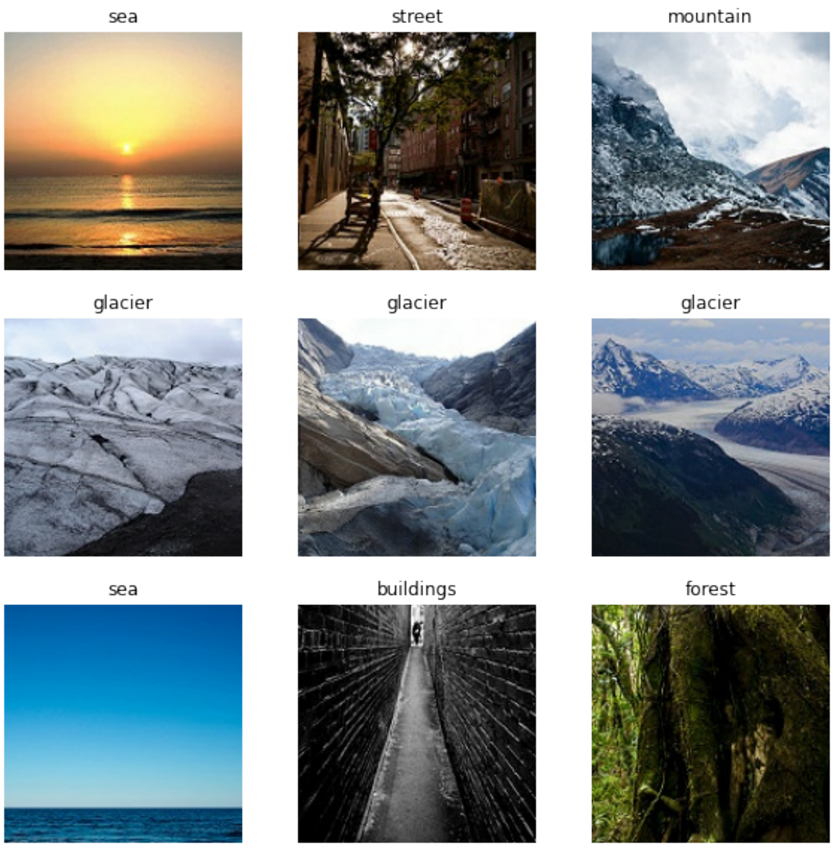
- Train with 14034 images
- Test with 3000 images
- Prediction with 7301 images

This data was originally published on https://datahack.analyticsvidhya.com by Intel for the Image Classification Competition.

Data Visualization

- Data visualization is required to better understand what kind of images will be used in the project, as well as to detect some patterns and/or difficulties that may occur during the workflow.
- On the sample of training images (*Figure 1*), it can be seen that there can be difficulties in predicting street and building classes, since images of buildings often contain street elements, and images of streets often contain buildings on them. The same is true for images of mountains and glaciers. It can be assumed that more accurate predictions will have the sea and forest classes.

Figure 1: Sample of Training Images



Building Input Pipeline

Data Performance Configuration

Buffered prefetching will be used to make sure to get data from disk without having I/O become blocking. These two important methods should be used when loading data:

- **Dataset.cache** keeps the images in memory after they're loaded off disk during the first epoch. This will ensure the dataset does not become a bottleneck while training the model. If the dataset is too large to fit into memory, this method also can be used to create a performant on-disk cache.
- **Dataset.prefetch** overlaps data preprocessing and model execution while training.

Data Standardization

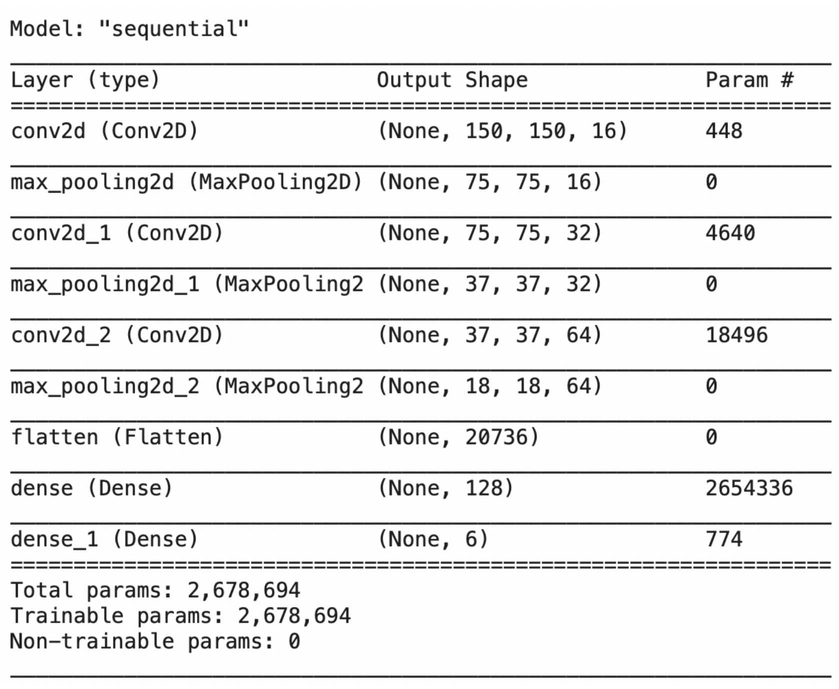
The RGB channel values of the images are in the [0, 255] range. This is not ideal for a neural network; in general, input values should be small. Using Rescaling layer, the values will be multiplied by (1/255) and thus standardized to the range [0, 1].

Modeling

Simple CNN Model (sCNN)

- The Sequential model consists of three convolution blocks with a max pooling layer in each of them. There's a fully-connected layer with 128 units on top of it, which is activated by a ReLU activation function. This model is a simple model and has not been tuned for high accuracy.
- The Compile method configures the model for training and validation using the optimizer, loss function, and evaluation metrics. All the models in this workflow will use the Adam optimizer, the Sparse Categorical Cross-entropy loss function, and the Accuracy evaluation metric.
- Model summary (*Figure 2*) shows all layers of the neural network.

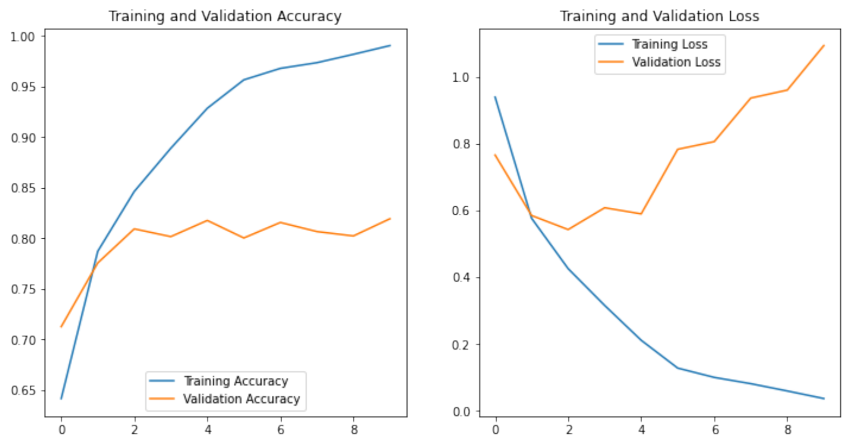
Figure 2: Model Summary



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 150, 150, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 75, 75, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 75, 75, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 37, 37, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 37, 37, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 18, 18, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 20736)             0
_________________________________________________________________
dense (Dense)                (None, 128)               2654336
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 774
=================================================================
Total params: 2,678,694
Trainable params: 2,678,694
Non-trainable params: 0
_________________________________________________________________
```

- The model will be trained using 10 epochs, and the test dataset as the validation data. After training the model, the validation loss is 1.0929 and the validation accuracy is 81.93%.
- The plots (*Figure 3*) show that the training accuracy increases linearly over time, while the validation accuracy stops at about 80% during the training process. Also, the difference in accuracy between training and validation accuracy is noticeable — a sign of overfitting. When there are a small number of training examples, the model sometimes learns from noises or unwanted details from training examples — to an extent that it negatively impacts the performance of the model on new examples. This phenomenon is known as overfitting. It means that the model will have a difficult time generalizing on a new dataset. To deal with overfitting in the training process, the base model will be tuned with data augmentation, and a Dropout layer will also be added to the model.

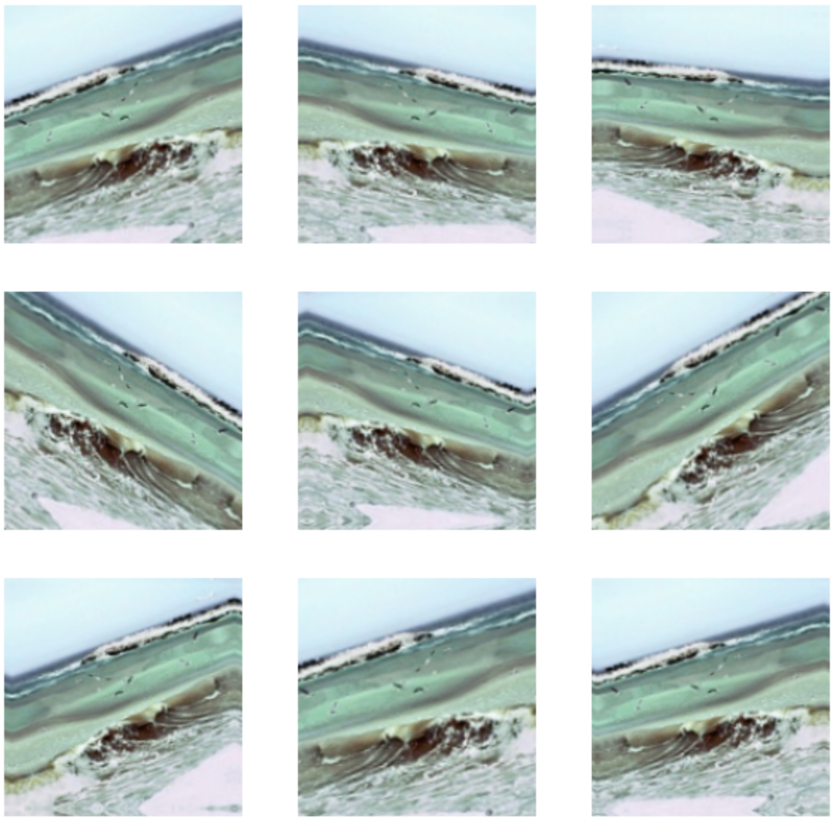
Figure 3: Model Accuracy and Loss



Augmented CNN Model (aCNN)

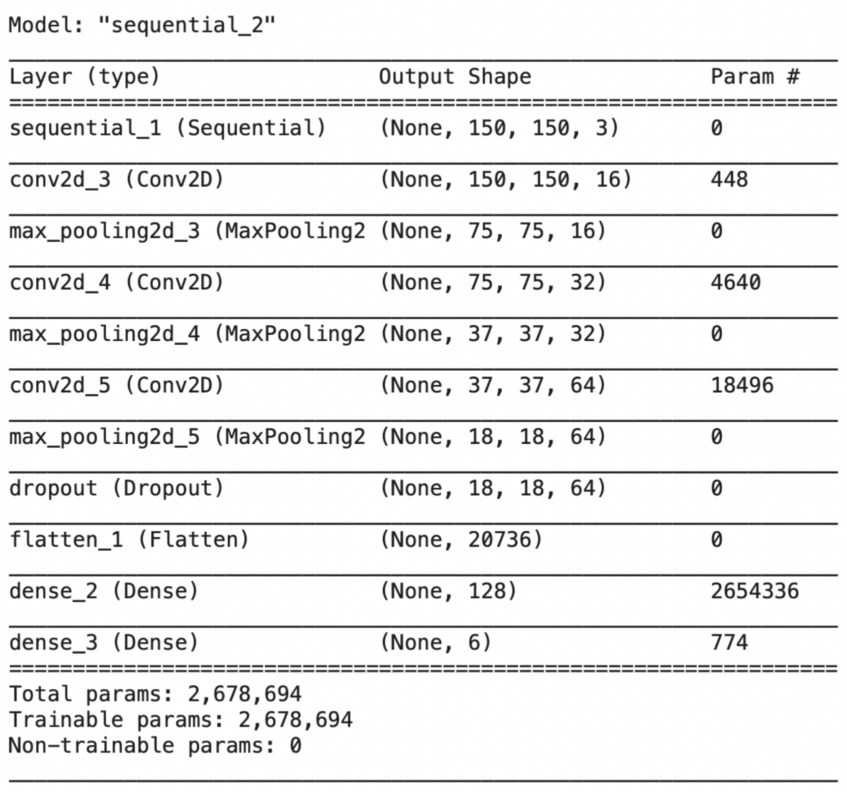
- Data augmentation takes the approach of generating additional training data from existing examples by augmenting them using random transformations that provide believable-looking images. This helps to expose the model to more data aspects and to generalize it better. Data Augmentation will be implemented using the RandomFlip, RandomRotation and RandomZoom layers. The augmented examples applied to the same image several times are shown in *Figure 4*.

Figure 4: Augmented Examples



- A new model will be created using the Data Augmentation approach and Dropout layer based on the previous baseline model. All the layers of the new model are shown in *Figure 5*.

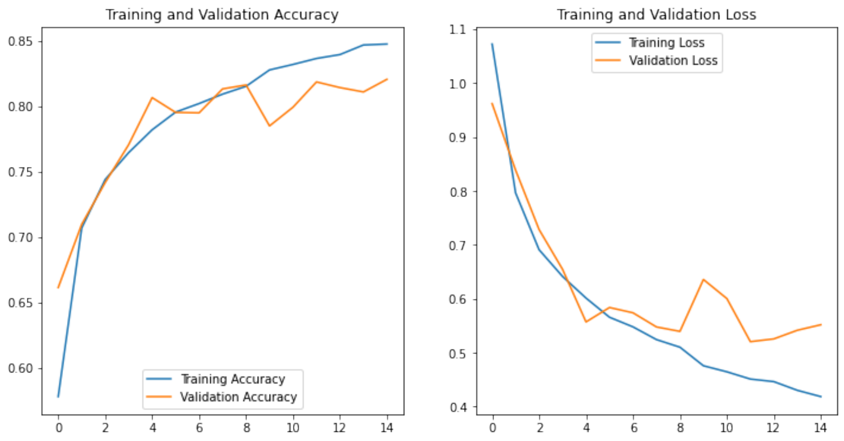
Figure 5: Model Summary



```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
sequential_1 (Sequential)    (None, 150, 150, 3)       0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 150, 150, 16)      448       
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 75, 75, 16)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 75, 75, 32)        4640      
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 37, 37, 32)        0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 37, 37, 64)        18496     
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 18, 18, 64)        0         
_________________________________________________________________
dropout (Dropout)            (None, 18, 18, 64)        0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 20736)             0         
_________________________________________________________________
dense_2 (Dense)              (None, 128)               2654336   
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 774       
=================================================================
Total params: 2,678,694
Trainable params: 2,678,694
Non-trainable params: 0
_________________________________________________________________
```

- The model will be trained using 15 epochs, and the test dataset as the validation data. After training the augmented model, the validation loss is 0.5518 and the validation accuracy is 82.07%.
- The plots (*Figure 6*) show that implementing the data augmentation approach and adding the Dropout layer to the model practically didn't increase the accuracy, but reduced the loss by more than half. However, the accuracy is still below 85%. This result might be due to the fact that a very simple model was used in this workflow. Another, more complex model based on the pre-trained VGG16 model will be created to improve the accuracy and reduce the loss even more.

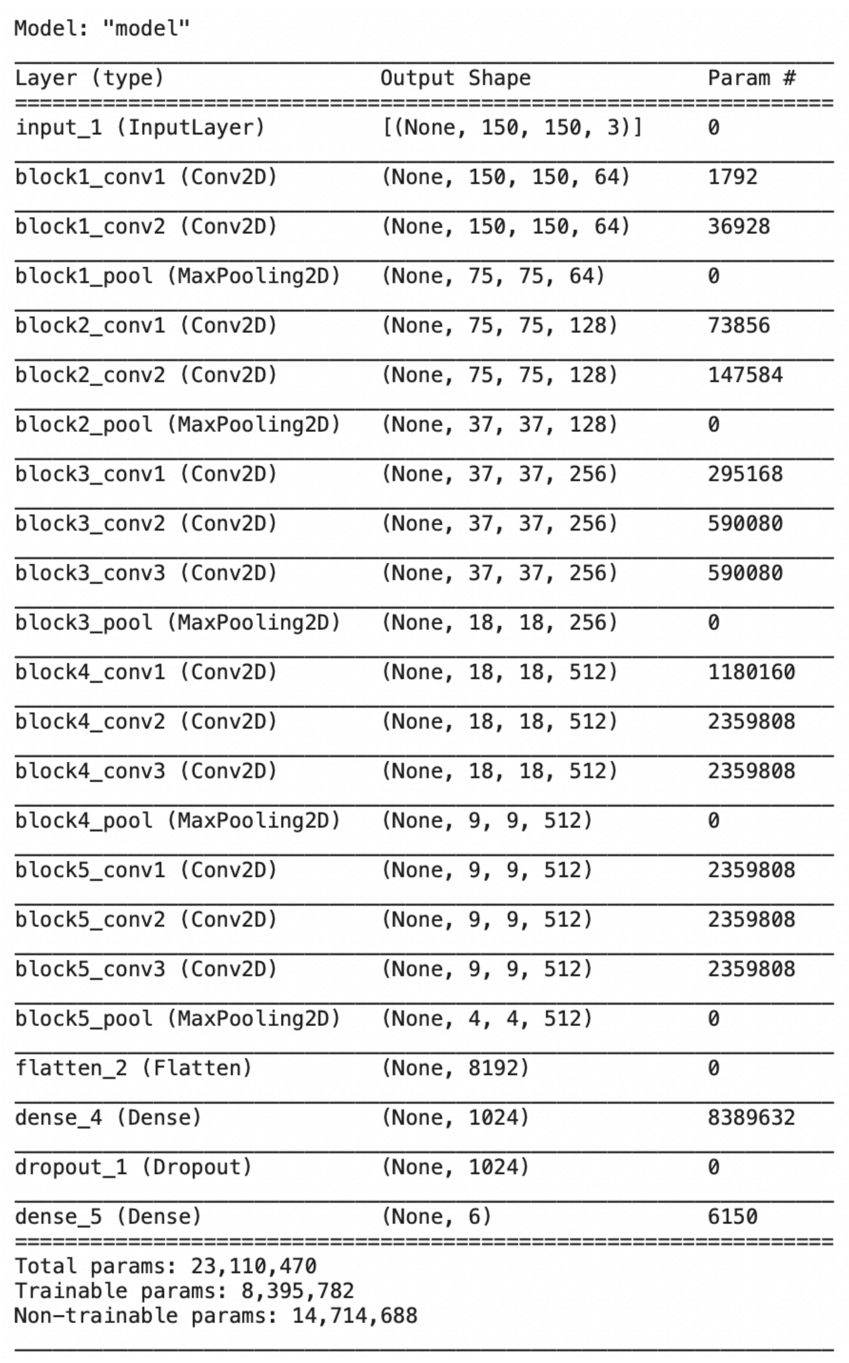
Figure 6: Model Accuracy and Loss



Complex CNN Model based on Pre-trained VGG16 Model (VGG16)

- First, a pre-trained VGG16 model will be loaded and then its layers will be defined as untrainable. Next, a fully-connected layer with 1024 units on top of it, which is activated by a ReLU activation function and Dropout layer will end up the model. All the layers of the complex model are shown in *Figure 7*.

Figure 7: Model Summary

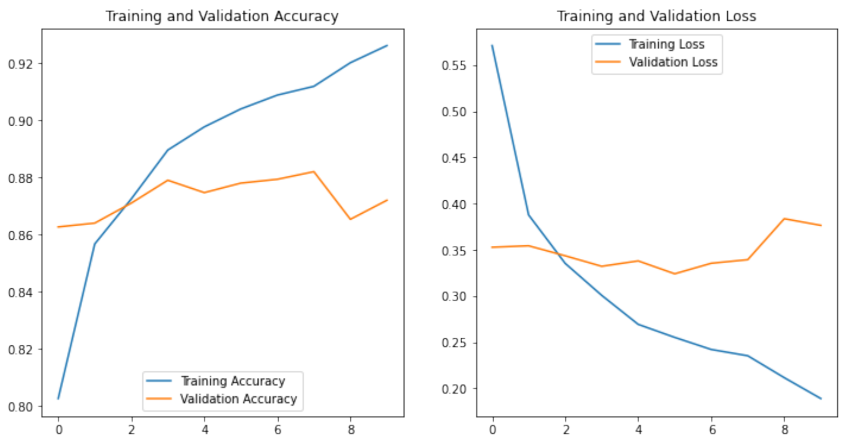


```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 150, 150, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 150, 150, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 150, 150, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 75, 75, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 75, 75, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 75, 75, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 37, 37, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 37, 37, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 18, 18, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 18, 18, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_4 (Dense)              (None, 1024)              8389632
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_5 (Dense)              (None, 6)                 6150
=================================================================
Total params: 23,110,470
Trainable params: 8,395,782
Non-trainable params: 14,714,688
_________________________________________________________________
```

- The model will be trained using 10 epochs, and the test dataset as the validation data. After training the complex model, the validation loss is 0.3764 and the validation accuracy is 87.2%.
- The plots (*Figure 8*) show that using the pre-trained complex model significantly improves the result — the accuracy increased by 5%, and the loss decreased almost 3 times, compared to the baseline model. Unfortunately, there is also a sign of overfitting this time. This indicates that the model needs

fine-tuning of the parameters to deal with overfitting and obtain higher accuracy.

Figure 8: Model Accuracy and Loss



Model Comparison

The training results of all 3 models (*Figure 9*) show that the accuracy of the first two models is almost the same, while the loss in the second model is reduced by half. While the second model is a more advanced version of the first simple model, it is still a very simple model for accurate prediction. And this can be clearly seen in the results of the third complex model, where the accuracy has significantly increased compared to the first two models, and the loss has significantly decreased. Therefore, a third, more complex and more accurate model, based on the pre-trained VGG16 model, will be used for the prediction of new unlabeled images.

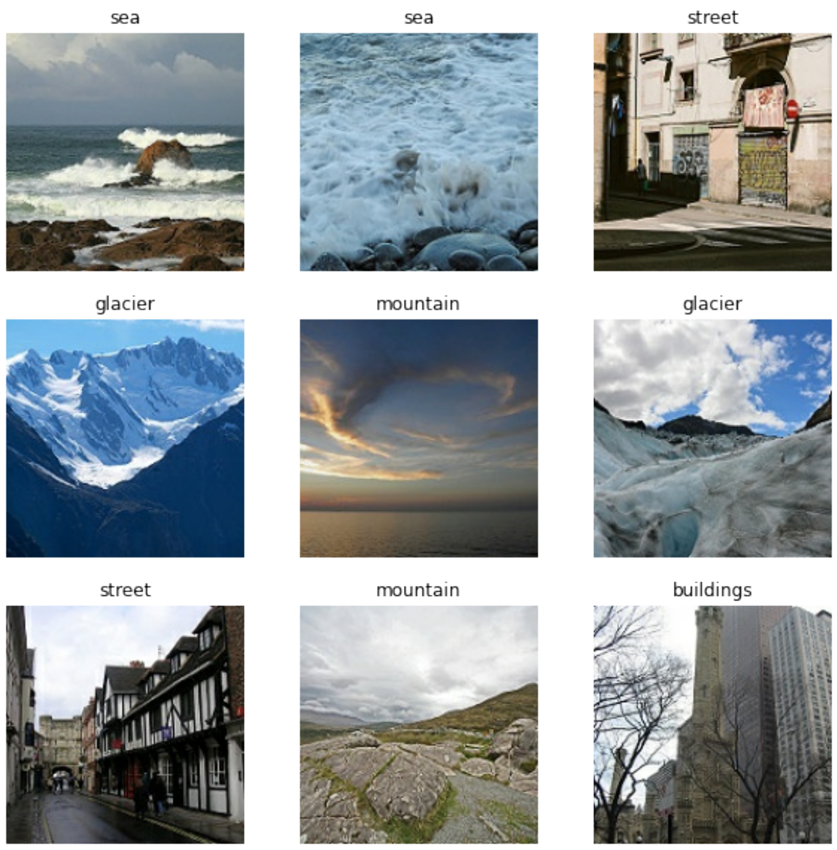
Figure 9: Model Results

	Loss	Accuracy
sCNN	1.092877	0.819333
aCNN	0.551755	0.820667
VGG16	0.376446	0.872000

Prediction

Results of the prediction performed on new unlabeled data using a complex model based on the pre-trained VGG16 model are shown as a sample of the predicted images (*Figure 10*). They confirm the assumptions that images with buildings, streets, mountains and glaciers on them will be poorly classified. But, this is a weakness not as much of the model itself, as of the input data.

Figure 10: Sample of Predicted Images



Key Findings and Insights

- Analyzing the images from the training dataset, it can be noticed that many of them are labeled incorrectly.
- The classes that are mostly difficult to classify are **streets** and **buildings**, and **mountains** and **glaciers**.
- A basic model with no tuning or optimization is able to provide good accuracy rate (~80%).

- Implementing the data augmentation approach and adding the Dropout layer provide slightly better results than the basic model, and also deal with overfitting.
- On the other side, the tuned model with the data augmentation approach and Dropout layer provides the same results as the baseline model trained in 3 epochs.
- The complex pre-trained VGG16 model provides the best results, but it also needs additional tuning.

Next Steps

- One way to increase performance even further is to train (or "fine-tune") the weights of the top layers of the pre-trained model alongside the training of the added classifier. The training process will force the weights to be tuned from generic feature maps to features associated specifically with the dataset. In most convolutional networks, the higher up a layer is, the more specialized it is. The first few layers learn very simple and generic features that generalize to almost all types of images. As you go higher up, the features are increasingly more specific to the dataset on which the model was trained. The goal of fine-tuning is to adapt these specialized features to work with the new dataset, rather than overwrite the generic learning.
- Another way is to use other models to train the data, such as InceptionV3 and ResNet50, before fine-tuning the VGG16 model. This will allow to select the best model for further fine-tuning and more accurate prediction on the new data.

Appendix

GitHub URL:
https://github.com/evgenyzorin/IBM-Machine-Learning/tree/main/Deep-Learning