Report on Intel Image Classification Using Convolutional Neural Networks

Introduction

This project focuses on classifying images into six distinct categories: buildings, forest, glacier, mountain, sea, and street. The aim is to develop a Convolutional Neural Network (CNN) model that accurately classifies images according to these predefined classes, serving as an illustrative example of supervised machine learning applied in computer vision.

Dataset Overview

The dataset used for this project is structured into three main components:

- **Training Set**: Located in seg_train, containing subfolders for each category with a total of 7,070 images.
- **Testing Set**: Located in seg_test, with a similar structure to the training set, containing 1,120 images.
- **Prediction Set**: Located in seg_pred, containing images that are to be classified without any corresponding labels.

Sample data

Data Exploration

The initial phase of the project involves exploring and visualizing the dataset:

- **Image Count**: The code iterates through each folder of the training and testing datasets to count the number of images, providing insights into the dataset distribution.
- **Image Size Analysis**: The dimensions of images in the training, testing, and prediction sets are examined, ensuring consistency in image sizes (all images are resized to 100x100 pixels).

Example of Image Count

For the training dataset:

- Buildings: 1,200 images
- Forest: 1,200 images
- Glacier: 1,200 images
- Mountain: 1,200 images
- Sea: 1,200 images
- Street: 1,200 images

For the testing dataset, similar counts were recorded.

Data Preparation

Images were processed to ensure uniformity:

- Each image was resized to 100x100 pixels using OpenCV.
- Labels for the training and testing datasets were encoded into a categorical format using a dictionary for mapping.

Encoding Example python

```
code = {'buildings':0 ,'forest':1,'glacier':2,'mountain':3,'sea':4,'street':5}
def getcode(n) :
    for x , y in code.items() :
        if n == y :
            return x
```

Model Architecture

The CNN model architecture consists of several layers designed for feature extraction and classification:

1. Convolutional Layers: Multiple layers using Leaky ReLU activations to capture spatial features.
2. Batch Normalization: Applied after convolutional layers to stabilize and accelerate training.
3. Max Pooling: Reduces dimensionality, allowing the model to focus on the most important features.
4. Dropout Layers: Introduced to prevent overfitting by randomly setting a fraction of the input units to 0 during training.
5. Global Average Pooling: Condenses spatial dimensions to provide a compact representation before the final classification.
6. Dense Layers: Fully connected layers leading to an output layer with softmax activation for multi-class classification.

Model Summary

The model has the following layers:

- Input Layer: (100, 100, 3)
- Convolutional layers with increasing filter sizes (32, 64, 128, 256, 512).
- Dense layers culminating in an output layer with 6 units (one for each class).

```
 Total params: 13,268,678 (50.62 MB)
 Trainable params: 13,264,710 (50.60 MB)
 Non-trainable params: 3,968 (15.50 KB)
None
```

Model Compilation

- The model is compiled with:
- Optimizer: Adam, with a learning rate of 0.001.
- Loss Function: Sparse categorical cross-entropy.
- Metrics: Accuracy.

Data Augmentation

To improve model robustness and generalization, data augmentation techniques were employed:

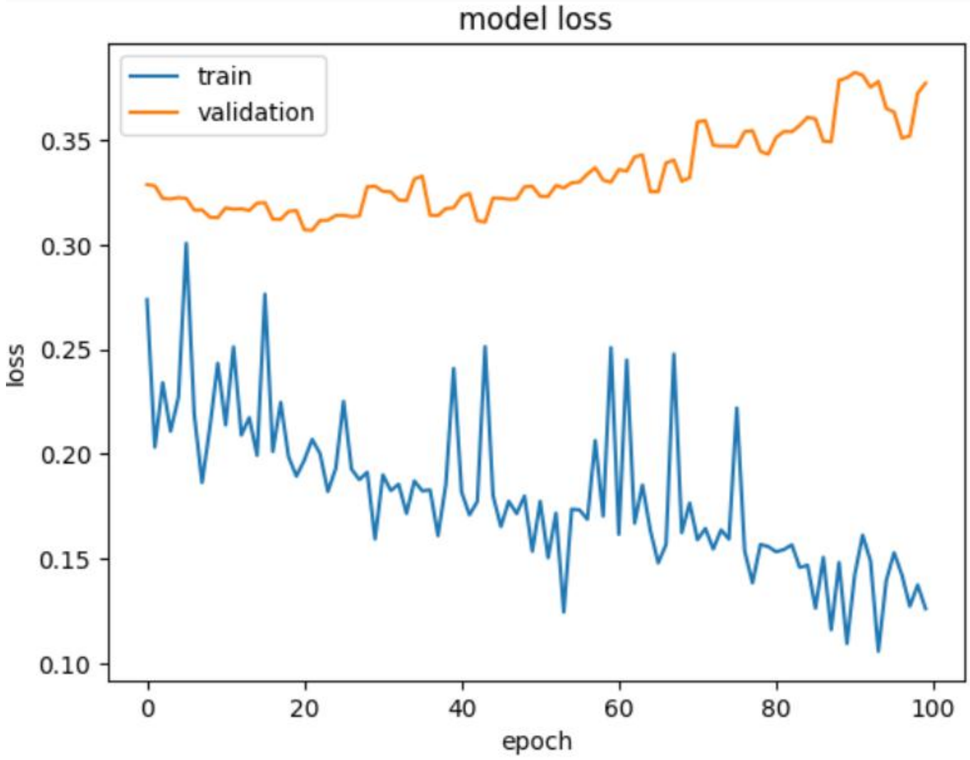
- Random rotations, zooms, shifts (width and height), and scaling.

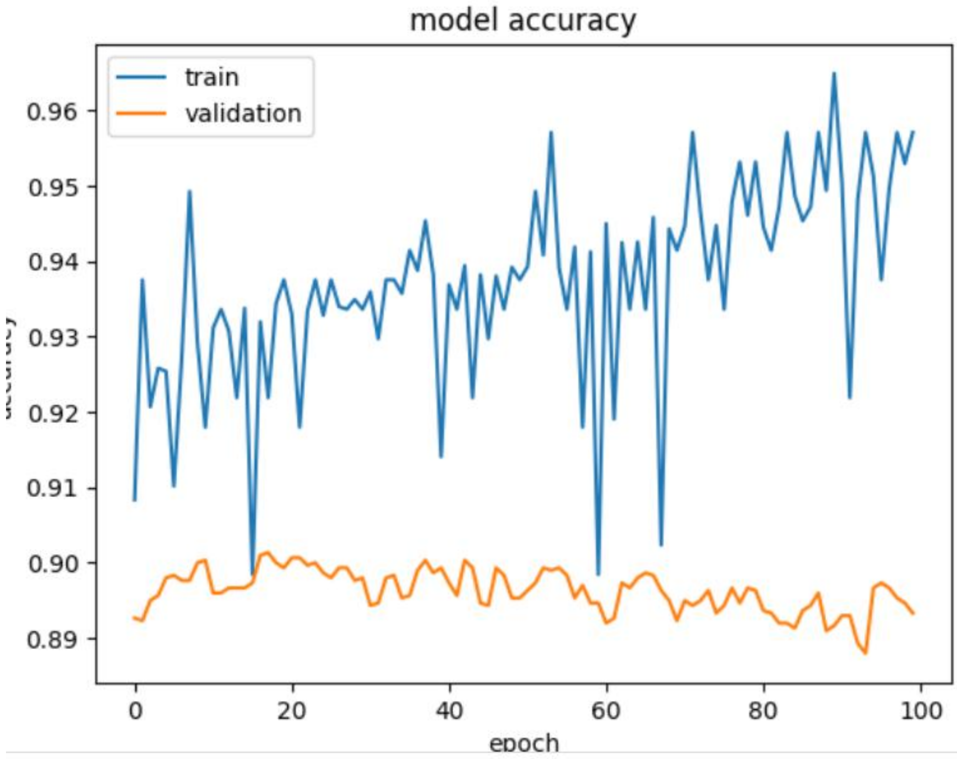
Training

The model was trained for 2 epochs with a batch size of 256. Early stopping and learning rate reduction were used to optimize the training process.

Training History

The training history includes metrics for accuracy and loss, which were plotted to visualize model performance over epochs 100.





Model Evaluation

The model was evaluated on the test dataset, yielding the following results:

Test Loss: .377

Test Accuracy: .893

```
model_loss,model_accuracy=model.evaluate(x_test,y_test)
print(f'test loss is  : {model_loss}')
print(f'test accuracy is : {model_accuracy}')
```

```
94/94 ━━━━━━━━━━━━━━━━━━━━ 3s 31ms/step - accuracy: 0.8880 - loss: 0.4202
test loss is  : 0.3772414028644562
test accuracy is : 0.8933333158493042
```

Predictions

Function show image

```
def predict_image_show(image):
    org_image = image
    single_image = np.expand_dims(org_image, axis=0)
    predictions = model.predict(single_image)
    predict_class=np.argmax(predictions,axis=1)
    plt.imshow(image ,cmap='gray')
    plt.title(f'Predicted Class: {getcode(predict_class[0])}')
    plt.axis('off')
    plt.show()
```

```
predict_image_show(x_pred[0])
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
```

Predicted Class: forest

```
predict_image_show(x_pred[900])
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```

Predicted Class: glacier

```
predict_image_show(x_pred[1400])
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
```

Predicted Class: street

Conclusion

The CNN model exhibits reasonable performance on the Intel Image Classification dataset. Key findings and suggestions for future work include:

Increase Epochs: Extend training time for potentially better results.

Advanced Data Augmentation: Experiment with more sophisticated techniques to enhance training data diversity.

Model Experimentation: Explore different architectures or leverage transfer learning from pre-trained models.