Exploring Vision-Based Models for Land-Usage Classification Using Remote Sensing Imagery Data

Presented By - Group 163
Anirudh Kaluri (akaluri)
Rishi Singhal (rsingha4)
Yazhuo Gao (ygao46)

Motivation - Why Land Use Classification?

- Understanding land-use patterns is crucial for sustainable resource management, urban planning, and environmental conservation.

- When combined with deep learning, these images offer insights for disaster recovery, resource allocation, precision agriculture, biodiversity monitoring, and infrastructure planning. A core challenge is accurately classifying land-use patterns from satellite imagery.

Problem Statement

Enhance the understanding of deep learning (DL) models' usefulness for land-use classification using three vision-based neural networks to classify remote sensing images

Dataset Used

- To explore the task of land-use classification, we use the UC-Merced Land-Use Dataset available at Kaggle [1].

- It contains satellite images of different urban regions in the US extracted from USGS National Map Urban Area Imagery collection.

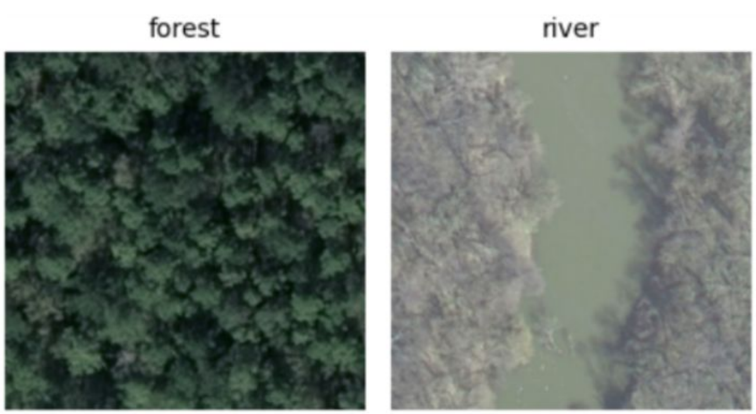
Input & Output of the Task

- *Input:*

  A RGB satellite image representing a region of land use, e.g., a forest. The majority of image size is 256x256.

- *Output:*

  A predicted class label (e.g., forest, river) corresponding to the land-use image feeded to the network.



Prior Work

- Rishi has previously worked with vision based models for different classification tasks.

Methodology

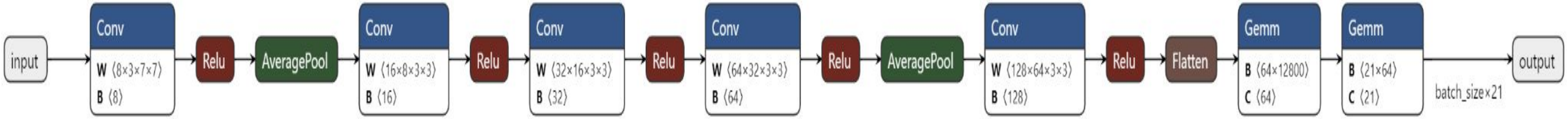
Data Splitting & Preprocessing

- *Data splitting* - We create a DataLoader which splits train set and validation set with 75% and 25%.
- *Data normalization* - We resize all images to 256x256 to maintain consistency.
- *Transformations* - We try multiple combinations of transformations to ensure better training performance.

```
Transform_pipeline = transforms.Compose([
    transforms.Resize((256, 256)),  # Resize images
    transforms.RandomHorizontalFlip(p=0.5),  # Flip images horizontally with 50% probability (74%)
    transforms.RandomRotation(degrees=30),  # Rotate images randomly within ±30 degrees (74%)
    # transforms.RandomResizedCrop(256, scale=(0.8, 1.0)),  # Crop images randomly (68%)
    # transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),  # Adjust brightness, contrast, etc. (65%)
    # transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)),  # Apply random translation (72%)
    # transforms.RandomPerspective(distortion_scale=0.2, p=0.5),  # Apply random perspective transformations (69%)
    transforms.ToTensor(),  # Convert image to tensor
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  # Normalize images
])
```

Baseline CNN Model

- *CNN Model* - We use a CNN consists of 5 2D-convolution layers with BatchNorm, ReLU, and AvePool. A classifier with dropout is followed at last to handle the features and output (21x1 size). Below is the architecture diagram:



Hyper-parameters & Metrics

- *Hyperparameters*

Dropout	Learning Rate	Weight Decay	Epochs	Loss function	Optimizer
p = 0.3	3e-4	1e-4	100	Cross Entropy Loss	SGD/Adam

- *Evaluation Metrics*

Accuracy	**Precision**	**Recall**	**F1-score**

Since our dataset is fully balanced, accuracy is considered to be the optimal choice to evaluate our model. We also plot **confusion matrix** on the evaluation set. Hence, providing a thorough metrics evaluation in our project.

Results

Analysis Across Multiple Data Transformations

Data Transformation	Accuracy
Random Horizontal Flip	74%
Random Rotation	74%
Random Resized Crop	68%
Color Jitter	65%
Random Affine	72%
Random Perspective	69%

We choose **Random Horizontal Flip and Rotation** as final data transformation techniques.

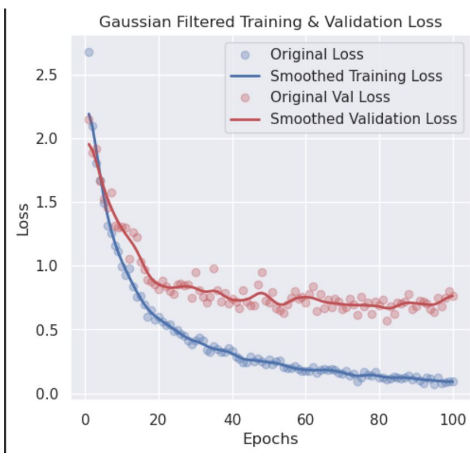
Adam V/S SGD Optimizer Analysis

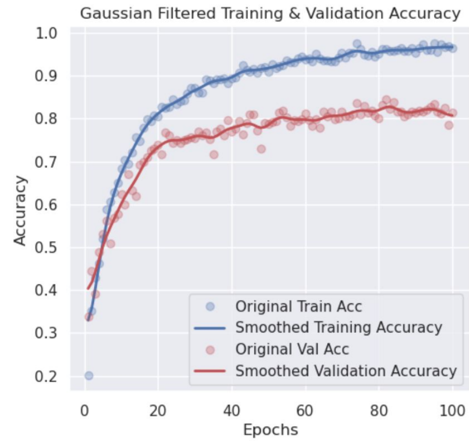
Optimizer	Accuracy	Precision	Recall	F1 Score
SGD	83.05%	83.14%	83.65%	82.66%
Adam	80.38%	80.93%	80.99%	80.21%

Using the best 2 data transformations, we analyse which optimizer is best - **SGD performs the best for our task!**

Loss-Accuracy Curves and Confusion Matrix

Using the best hyperparameters we plot the loss-accuracy curves and the confusion matrix (SGD + 2 data transformations + other parameters as discussed previously)





Confusion Matrix (Validation Set)

True Label \ Predicted Label	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	22	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	29	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	1	0	0
2	0	0	20	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	30	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
4	0	0	0	0	19	0	1	0	1	0	0	0	0	1	0	0	0	1	0	1	0
5	0	0	0	0	0	19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	1	0	9	0	0	0	0	1	5	0	0	0	0	0	0	0	2
7	1	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	1	0	0	0	21	1	0	0	0	1	0	0	0	1	1	0	0
9	0	0	0	0	0	0	0	0	0	19	0	0	1	0	0	0	2	0	0	0	0
10	1	0	0	0	1	0	0	0	0	0	27	0	0	0	0	0	0	0	0	1	0
11	0	0	0	0	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	7	0	0	0	0	0	17	5	0	0	1	0	0	0	1
13	0	0	0	0	0	0	1	0	0	0	0	0	1	22	0	0	0	0	0	0	0
14	0	0	0	0	1	0	0	0	3	0	0	0	0	0	18	0	0	0	1	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	26	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	20	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24	1	0	0
18	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	1	0	19	0	1
19	0	0	1	0	4	0	0	0	0	2	0	0	0	1	1	0	0	3	2	14	0
20	0	0	2	0	1	0	0	1	0	0	0	0	0	1	0	0	2	0	5	1	19

Accuracy - 83.05%, Precision - 83.14%, Recall - 83.65%, F1-score - 82.66%

Note: Since it is overfitting at later epochs we provide metrics for the epoch where we get the best validation accuracy.

Performance Time and Model Size

- Total Trainable Parameters: 920,469
- Model Size: 10.89 MB
- Per Epoch Train Time - 1.6 secs

The model was trained on a NVIDIA A100 GPU.

References

[1] UC Merced Land Use Dataset. Available:
https://www.kaggle.com/datasets/abdulhasibuddin/uc-merced-land-use-dataset/data.