Assignment #4 Satellite Image Classification Using Transfer Learning

P68131509 林均有

Explain that why to shuffle the image dataset and balance the labels?

Shuffling the image dataset ensures that the model doesn't memorize the order of the data, preventing overfitting and helping it generalize better. It also ensures that each mini-batch contains a diverse mix of examples, leading to more stable and effective learning.

Balancing the labels ensures that each class has an equal representation in the dataset, preventing the model from being biased toward the majority class. This helps the model perform better across all classes, especially the minority ones, and provides a fair evaluation of its performance.

Briefly describe the Model #1-#8 in the code, and show the performances of them

Model #1 (M1): Basic Fully Connected Neural Network

Model #1 is a simple fully connected neural network (also known as a dense network), where the image data is flattened and passed through a single hidden layer before producing an output using softmax activation. The model does not have any convolutional layers, which means it treats the input data as a vector rather than trying to learn spatial features. This model serves as a baseline, where the goal is to test the basic performance of a neural network on the image classification task, without leveraging complex techniques like convolutions or pre-trained models.

Model #2 (M2): Deep Fully Connected Network

Building upon the simplicity of M1, Model #2 introduces an additional hidden layer with 256 units. Both the hidden and output layers use ReLU and softmax activations, respectively. This model represents an attempt to improve performance over the basic model by increasing its depth, allowing it to learn more complex patterns in the data. The extra layer provides additional capacity to learn more complex representations of the input data, which could lead to better generalization on unseen examples compared to M1.

Model #3 (M3): Deep Fully Connected Network with Dropout

Model #3 takes the architecture from M2 and adds regularization through the use of dropout layers. Dropout is a technique where, during training, random units in the network are "dropped" (set to zero) to prevent overfitting. By introducing multiple dropout layers between hidden layers, M3 is able to generalize better, especially when the model is at risk of overfitting due to the increased number of hidden units. This model helps to balance the capacity to learn complex features while avoiding overfitting to the training data.

Model #4 (M4): Convolutional Neural Network (CNN)

Model #4 introduces convolutional layers, making it the first CNN in this series. The model starts with two convolutional layers followed by max-pooling layers, which are designed to automatically learn spatial hierarchies of features in the images. Convolutional layers help detect patterns such as edges and textures, which are critical for image classification. This model marks the transition from fully connected networks to architectures better suited for images, and it is expected to perform significantly better on image classification tasks due to its ability to extract meaningful spatial features.

Model #5 (M5): Deeper Convolutional Neural Network

Building upon the success of M4, Model #5 introduces even deeper convolutional layers. With four convolutional layers followed by max-pooling and dropout, this model increases the depth of the network, which allows it to learn more intricate features in the images. The more layers a CNN has, the better it can model complex patterns in the data. The dropout layers are used to avoid overfitting, making this model a stronger candidate for more complex image datasets, especially in tasks where subtle image features are important for classification.

Model #6 (M6): Transfer Learning with VGG16

Model #6 introduces transfer learning by using the VGG16 model as a pre-trained convolutional base. Instead of training convolutional layers from scratch, the pre-trained VGG16 network, which was trained on large datasets like ImageNet, is used to extract high-level features from the images. These features are then passed through a custom classifier consisting of a flattening layer and dense layers. By leveraging the power of a pre-trained model, M6 benefits from the learned features of a large dataset, significantly improving its performance on smaller datasets with less training time required.

Model #7 (M7): Transfer Learning with Data Augmentation

Model #7 takes M6 one step further by incorporating data augmentation. Data augmentation involves generating new training examples by applying random transformations to the original data, such as flipping and rotating the images. This increases the diversity of the training set, preventing overfitting and helping the model generalize better. Additionally, in M7, the VGG16 convolutional base layers are frozen initially, which helps the model focus on learning the new classification task before fine-tuning the convolutional base in later stages. This combination of transfer learning and data augmentation makes M7 a robust model for image classification tasks.

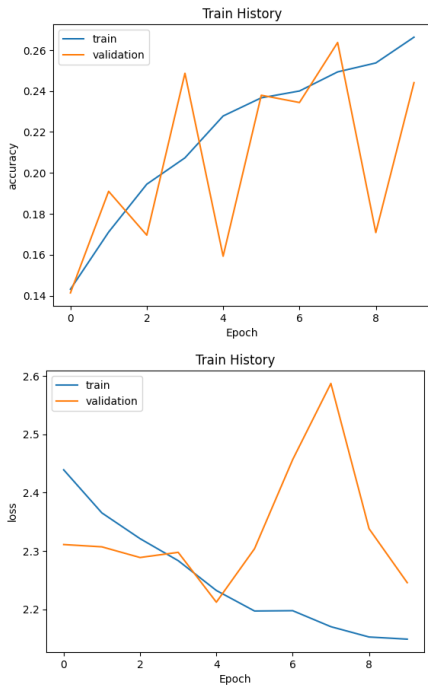
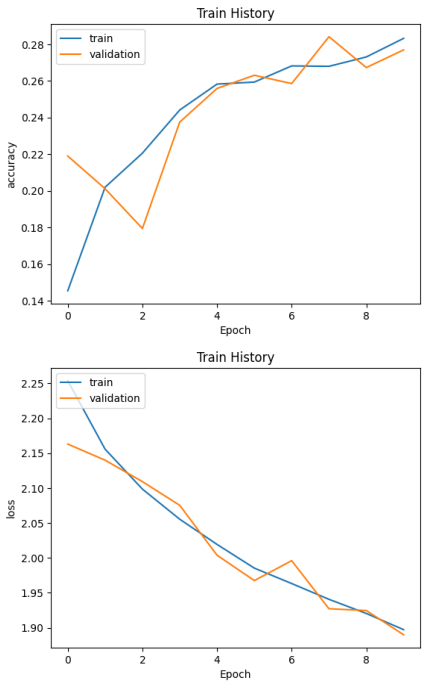
Model #8 (M8): Transfer Learning with Learning Rate Scheduling and Early Stopping

Model #8 combines the features of M7 with additional callbacks for better

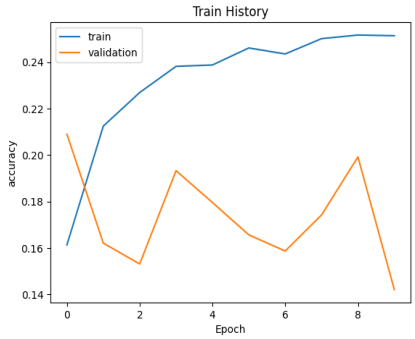
training efficiency. It uses early stopping, which halts training when the validation loss stops improving, and learning rate scheduling, which reduces the learning rate when the model plateaus. This model adapts to changes in training dynamics by lowering the learning rate as the model starts to converge, preventing the model from overshooting optimal weights. These callbacks, along with the transfer learning approach, result in a model that can efficiently train on image data while optimizing the learning process for better performance and faster convergence.

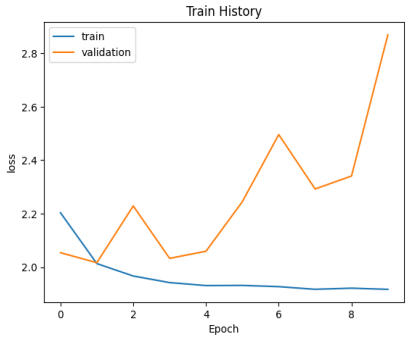
Each model in the series represents a progression in complexity, from basic neural networks to advanced techniques like transfer learning, data augmentation, and regularization, reflecting the evolution of deep learning practices for image classification.

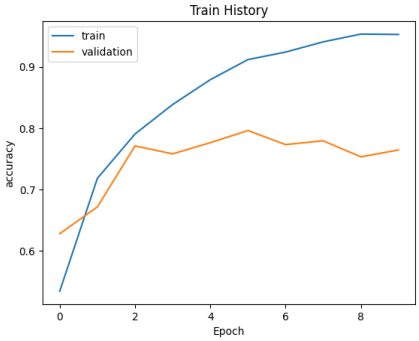
Performance:

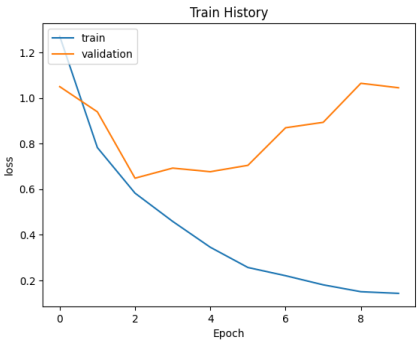
Model 1	Medel 2
	

Model 3	Medel 4

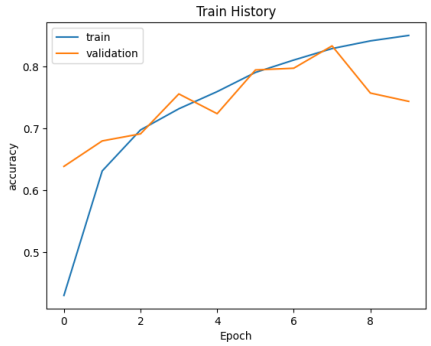


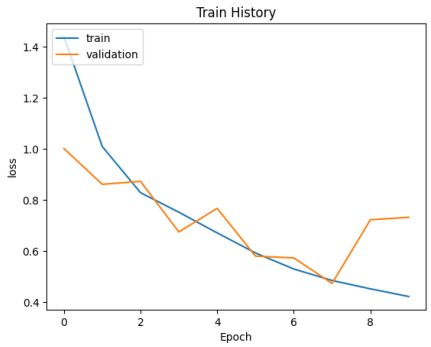


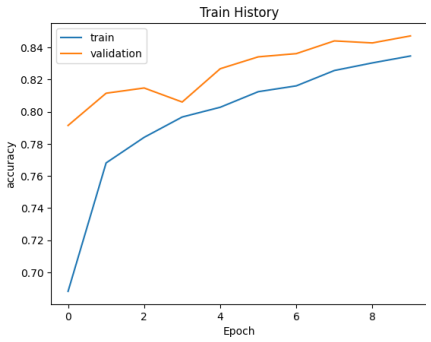


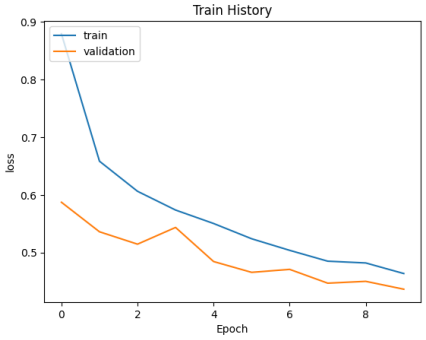


Model 5	Medel 6

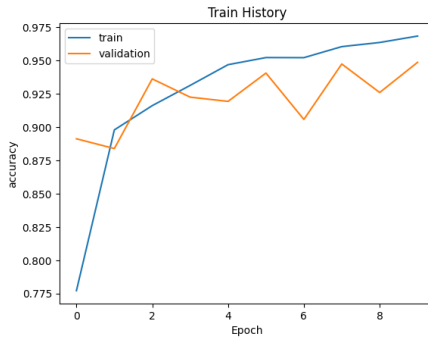


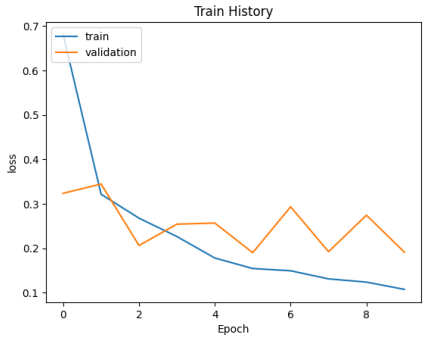


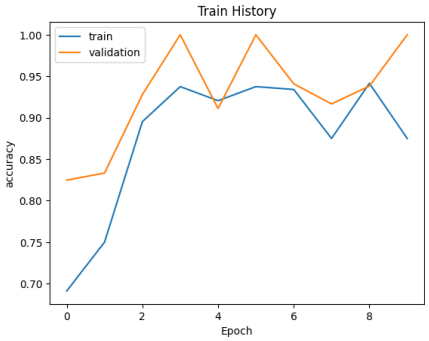


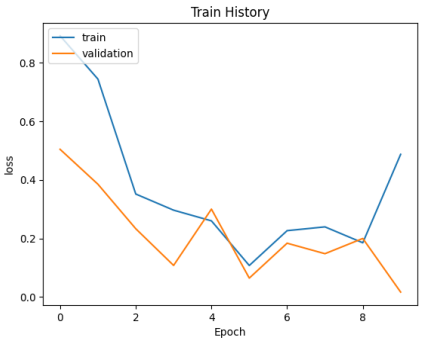


Model 7	Medel 8

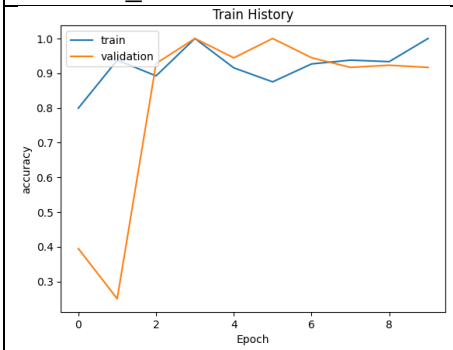


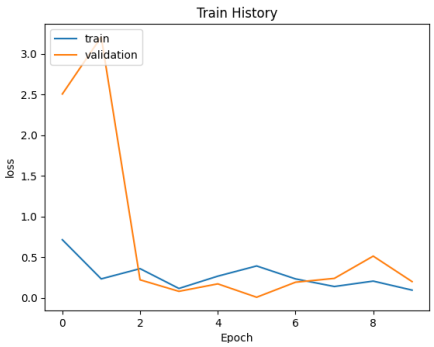


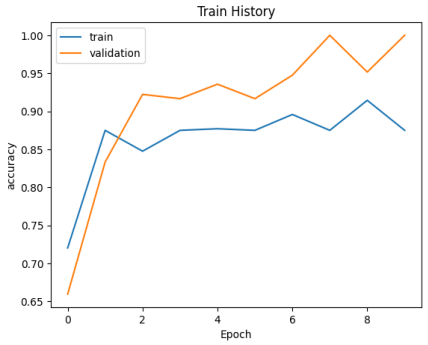


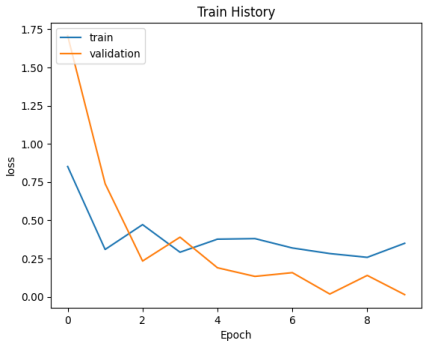


Model 8_ResNet50	Medel 8 EfficientNetB0

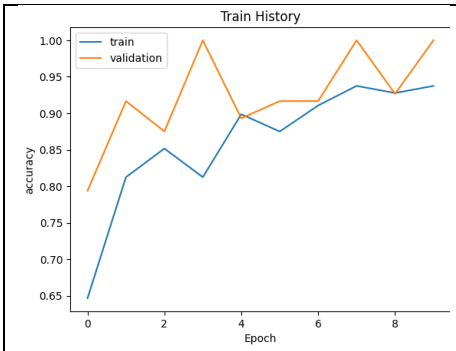


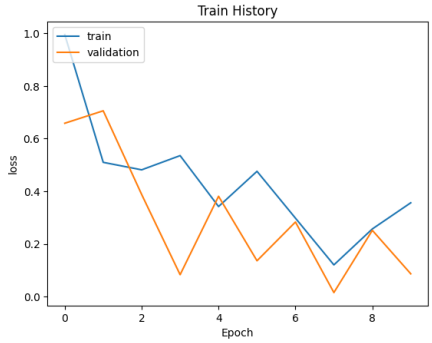


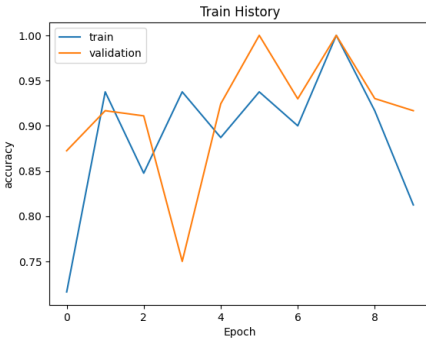


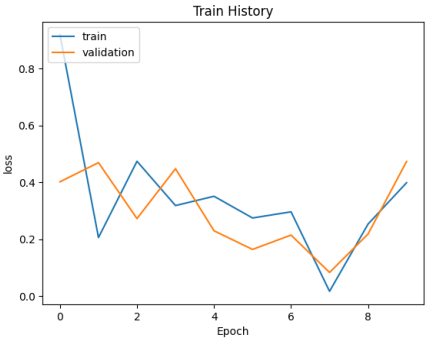


Model 8_VGG19	Medel 8 ResNet50v2

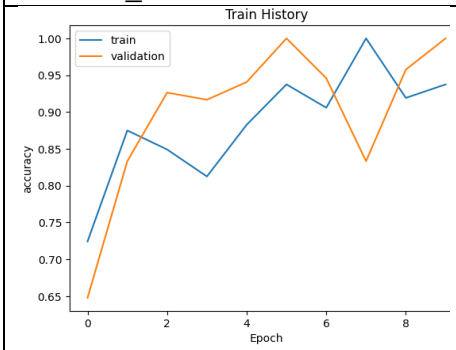


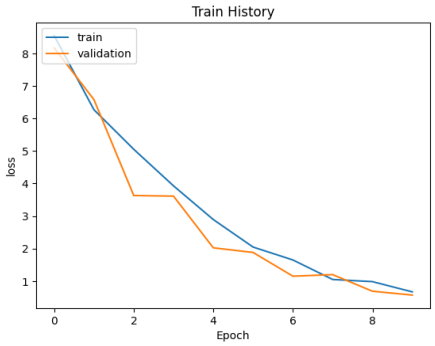




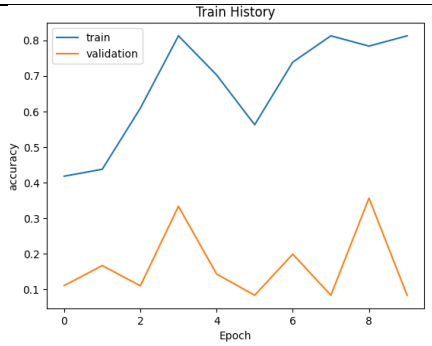


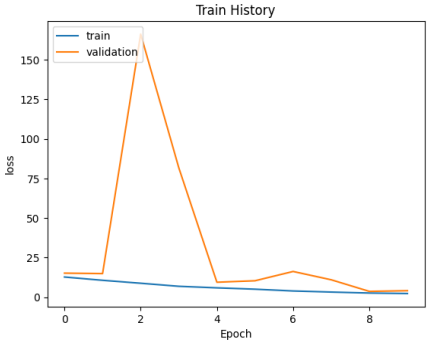
Model 8_ EfficientNetB0 + L2





Medel 8 EfficientNetB0 + L2 +Dense





y_true	b'AnnualCrop'	b'Forest'	b'HerbaceousVegetation'	b'Highway'	b'Industrial'
label_count	1500	1500	1500	1250	1250
M1_class_acc	0.198	0	0.442	0.3128	0.6248
M2_class_acc	0.191333	0.806667	0.043333	0	0.488
M3_class_acc	0	0	0	0	0
M4_class_acc	0.888667	0.922	0.579333	0.536	0.8296
M5_class_acc	0.921333	0.681333	0.58	0.7848	0.9784
M6_class_acc	0.918667	0.990667	0.907333	0.968	0.9896
M7_class_acc	0.93	0.97	0.898	0.9544	0.9576
M8_class_acc	0.961333	0.822667	0.943333	0.96	0.988

y_true	b'Pasture'	b'PermanentCrop'	b'Residential'	b'River'	b'SeaLake'
label_count	1000	1250	1500	1250	1500
M1_class_acc	0	0	0.009333	0.08	0.660667
M2_class_acc	0	0.092	0.562	0.076	0.196
M3_class_acc	0	0	1	0	0
M4_class_acc	0.703	0.7344	0.667333	0.52	0.934
M5_class_acc	0.476	0.36	0.812667	0.7832	0.962
M6_class_acc	0.956	0.9472	0.932667	0.916	0.966667
M7_class_acc	0.931	0.9352	0.906	0.9696	0.994667
M8_class_acc	0.955	0.896	0.944667	0.9112	0.992

Overall Accuracy	M1	M2	M3	M4	M5	M6	M7	M8
	0.240	0.261	0.111	0.738	0.744	0.949	0.944	0.940

Compare three different backbones in Model #8

M8 Overall Accuracy	Origin	VGG19	ResNet50	ResNet50 V2	Efficient NetB0	Efficient NetB0 + L2	EfficientNetB0 + L2 + Dense
	0.940	0.918	0.946	0.923	0.951	0.957	0.111

The performance of M8 highlights the impact of different backbone architectures and optimization techniques on model accuracy. The **baseline model** (EfficientNetB0) achieves a strong starting accuracy of **0.940**, demonstrating the power of modern architecture scaling. EfficientNetB0's balanced design effectively captures features

without excessive computational cost.

When **VGG19** is used as the backbone, the accuracy drops to **0.918**. Despite its depth, VGG19 lacks modern innovations like skip connections or efficient parameter scaling, making it less effective for this dataset. In contrast, **ResNet50** performs better (**0.946**) due to residual connections, which address gradient vanishing and enable deeper layers to learn effectively. However, the slightly modified **ResNet50 V2** scores lower (**0.923**), possibly due to differences in batch normalization and activation strategies, which may not align well with this specific dataset.

Further optimizations of EfficientNetB0 yield the best results. Adding **L2 regularization** improves accuracy to **0.951** by penalizing large weights and reducing overfitting.

Modify the example code to improve the classification accuracy as much as you can

I added L2 regularization to EfficientNetB0, which improved its accuracy from **0.951** to **0.957**, demonstrating the effectiveness of this technique. L2 regularization penalizes large weights in the loss function, encouraging the model to learn smoother and more generalized features. This helps reduce the risk of overfitting, which is especially important for deep models like EfficientNetB0.
The original EfficientNetB0 already achieved high accuracy, but there were likely minor overfitting issues, particularly if the model performed significantly better on the training set than on the validation set. Adding L2 regularization improved the model's generalization, allowing it to perform better on unseen data.
A combination of **L2 regularization** with **EfficientNetB0** boosts accuracy to **0.957**, showcasing the importance of increasing model capacity while maintaining regularization for generalization.

Adding a Dense layer to the EfficientNetB0 model caused a significant drop in accuracy, from **0.957 to 0.111**, likely due to architectural or training issues. The Dense layer may have disrupted the pre-trained features learned by EfficientNetB0, which is optimized for feature extraction. When adding new layers, careful adjustments to their size, activation function, and initialization are needed to avoid misaligning learned features, making convergence difficult.