Combining Vanilla Scene Classification with Image Segmentation for Improved Scene Classification

Matthew Kaplan
Stanford University
mkaplan1@stanford.edu

Avery Rogers
Stanford University
averyr@stanford.edu

Abstract

We perform scene classification on the ADE20K dataset using both raw images and segmentation masks. We try vanilla scene classification from raw images, scene classification from combined raw images and ground-truth segmentation masks, and scene classification from combined raw images and custom learned segmentation masks. We find that combining raw images with their segmentation masks theoretically improves scene classification accuracy, but our own custom segmentation masks do not achieve high enough pixel-wise accuracy to produce improved scene classification accuracy more broadly.

1. Introduction

Scene classification is a critical task for automated photograph and video annotation and other high-level tasks that require a semantic understanding of a given environment. As scenes become more cluttered and complex, classifying scenes becomes more difficult.

Vanilla scene classification works much like object classification: the raw pixels of an image are evaluated together to determine the scene label, using any mix of fully connected layers and convolutional neural networks. However, vanilla classification is less well suited to scenes with many objects that may appear at random locations in a picture, and those objects may be in random relation to each other. Learning to identify a complex scene thus requires some semantic understanding of the components that make up the scene.

We seek to improve scene classifications for complex scenes by combining vanilla scene classification and semantic segmentation for objects within the scene to produce more accurate class labels. Our work will employ the carefully annotated ADE20K dataset, which contains both scene labels and segmentation masks for training and testing [6]. We select the top 50 most common scenes in the dataset for simplicity and to make sure each class has sufficient training data. Our method benefits from being a relatively simple model that will be trainable on our provided computing power and within our project timeline.

Our goal is to achieve significantly better accuracy on classifying scenes from the ADE20K than we would achieve using vanilla scene classification described above. We hope to do so without significant costs to efficiency and memory.

Our procedure will involve four parts, each of which builds on the step before it:

1. We will produce a baseline vanilla scene classification algorithm using the ResNet-50 pretrained model [2]. Our input will be raw scene images and our output will be a class label.

2. We will concatenate the raw scene images with the ground-truth segmentation masks provided by the ADE20K dataset and run ResNet-18 on the concatenated, 4-channel pixel matrix as a baseline to see if adding the segmentation mask improves scene classification accuracy [2].

3. We will build our own semantic segmentation algorithm to convert our raw images into segmentation masks using ResNet-50 for pixel-wise training [2].

4. Finally, we combine steps 1 and 3 to mimic the concatenation process in step 2 using our own segmentation masks. We concatenate the raw images with our learned segmentation masks and then use ResNet-50 to perform scene classification on the concatenated input [2].

Together, these four steps provide us with both a theoretical, ground-truth understanding of the possible improvement to scene classification by using a segmentation mask combined with raw pixel values, as well as a full implementation of both semantic segmentation and vanilla scene classification in practice. We expect to see that concatenating the segmentation mask to the raw pixel input will improve

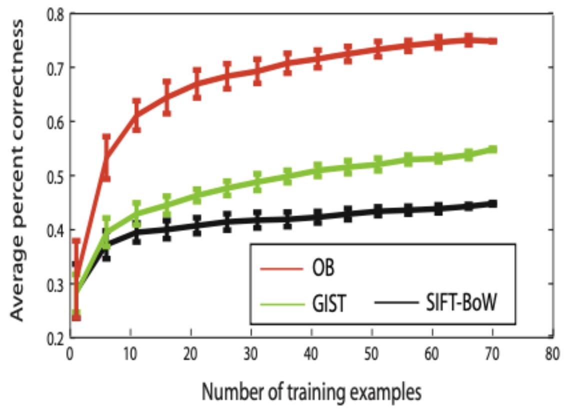

Figure 1. Percent of scenes correctly classified by Li et. al's object-detection model (”OB”) compared to GIST and SIFT-BoW, contemporaneous models for scene classification [3].

classification, and we hope our segmentation mask will perform similarly to the ground-truth segmentation mask used in step 2.

2. Related Work

One approach to improving scene classification accuracy is to identify objects within the scene first, and then to use those object classifications to help predict the broader scene class. This helps by adding a more discrete and achievable intermediate task between the low-level pixel values and the high-level scene class label. For example, a cluttered scene consisting of many identifiable boats might be easier to classify as ”ocean” or ”harbor” once the boats in the image are classified.

Object classification for scene classification has been attempted before, most notably by Li et al. (2012) [3]. Li et al. used a series of object detectors to scan over a scene and identify instances of various objects within the scene. By using objects to identify the broader scene, Li. et al. improved scene classification significantly compared to contemporaneous state-of-the-art models [3].

Another approach is to use image segmentation masks as an intermediary layer in between the raw pixel processing and the scene labelling steps. This approach was pioneered by Campbell et al. (1996) in their paper, ”The Automatic Classification of Outdoor Images” [1]. Campbell et al. used a K-Means algorithm to learn the segments of an outdoor image and then trained a neural network to learn the scene label from the segmentation mask [1].

These two approaches have led to a plethora of research on scene classification using object detection and segmentation masks. A prime example is Yao et al.'s (2012) approach that combines both object detection and segmentation masks, matching accuracy and improving speed over contemporaraneous state-of-the-art methods [5].

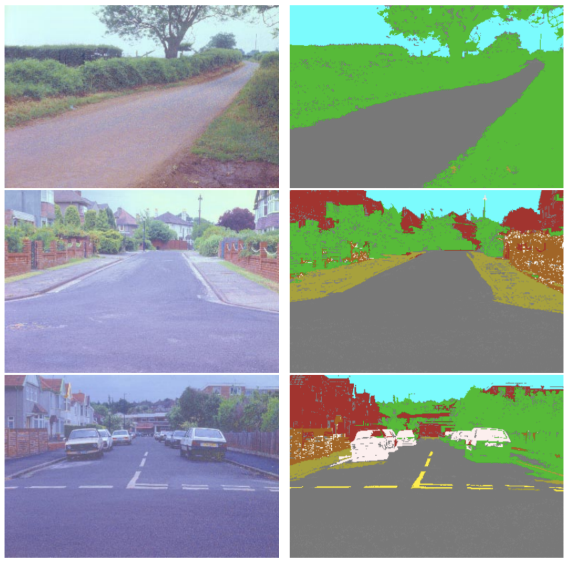

Figure 2. Campbell et al. used image segmentation for scene classification. Original images are on the left, and segmented images are on the right. [1]

3. Methods

3.1. Inputs

Our inputs are relatively straightforward. For Steps 1 and 3, we input arrays of raw images of shape `(N, 128, 128, 3)`, where N is the number of images, 128 is the side length of the input image (which we trim to be a small square because of RAM constraints), and 3 is the RGB channels that make up each pixel.

For Steps 2 and 4, we input our raw images concatenated with our segmentation masks and learn the scene labels from the concatenated input. Since the raw images array is of size `(N, 128, 128, 3)` and the segmentation mask array is of size `(N, 128, 128, 1)` (explained in next section), we concatenate along the third axis to get a final concatenated array of shape `(N, 128, 128, 4)`.

3.2. Outputs

In Step 1 (Vanilla Scene Classification), we expect our model to take in raw images and output a scene classification label, which was created in our preprocessing. The output size will be `(N, 50)`, where `N` is the first dimension of the input training, validation, or testing array and 50 is the number of classes we are predicting. The 2-dimensional output array thus contains, for each input image, a class score for each of our 50 classes.

In Steps 2 and 4 (Scene Classification from Raw Images and Segmentation Masks), our inputted raw image and

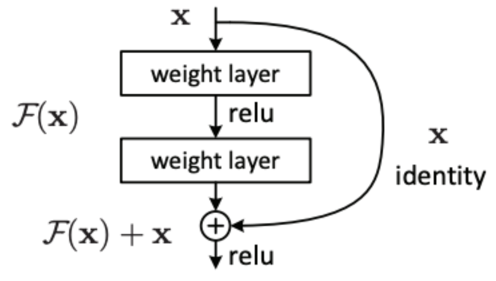
x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 3. A visualization of a residual learning building block from the ResNet class. [2]

segmentation mask concatenation will produce an array of scene classification labels as in Step 1. Like above, the output size will be `(N, 50)`, where `N` is the first dimension of the input training, validation, or testing array, and 50 is the number of classes. Again, the 2-dimensional output array contains the class scores for each input image.

In Step 3 (Creating Segmentation Masks), our input will be an array of raw images of size `(N, 128, 128, 3)`, and the model will output a tensor of size `(N, 128, 128, 253)`. Here, each channel represents an object class. The pixel values at each channel represent the class score at the corresponding image location.

In Steps 1, 2, and 4, the scene predictions (of shape `(N, 1)`) and in Steps 3 and 4 the predicted segmentation masks (of shape `(N, 128, 128, 1)`) are generated by taking the argmax across class scores of the model outputs.

3.3. Learning Algorithms

For each step in our process, we rely on the ResNet learning algorithms developed by He et al. [2]. The ResNet class of pretrained CNN algorithms relies on the concept of a residual learning building block, as illustrated in Figure 3. The residual building block is a tool used to improve optimization in large neural networks, which is traditionally the biggest barrier to high performance as networks become very deep. Each residual block is built from a 'plain' neural network path of two convolutional layers combined with a ReLU activation layer, producing some output $H(x)$. Rather than only defining $H(x)$ by the plain network structure, which we call $F(x)$, we will create a pathway from the input x to $H(x)$ that is simply an identity mapping. We will skip $F(x)$ entirely and simply use the identity mapping if it produces a better result. In this way, a neural network can be made very deep while having flexibility to skip layers that decrease performance. ResNet-50 combines many residual blocks into an overall architecture of 50 layers for flexible and easily optimized deep learning.

For Step 1 (Vanilla Scene Classification), we use the ResNet-50 model as our learning algorithm to achieve a strong baseline result. We chose ResNet-50 for its balance of depth and speed. Because ResNet-50 is 50 layers deep, it can achieve a high level of accuracy, but also utilizes a residual network structure that can operate within our RAM and disk space constraints and trains quickly enough for us to iterate and run many epochs. In order to make ResNet-50 compatible with our number of classes, we changed the fully connected layer at the end of ResNet-50 to output a length-50 vector of class scores for each image. We also used the Adam optimizer with our ResNet-50 network, chosen because the Adam optimizer is shown to perform best for most CNN tasks.

For Step 2 (Scene Classification from Raw Images and Ground-Truth Segmentation Masks), we tried multiple strategies for our learning algorithm. We first wrote a custom convolutional neural network with 24 layers, including a number of batch normalization, max pooling, and dropout layers. Our custom algorithm did not quite match the baseline from Step 1, though we predicted that it would outperform our results in Step 1 due to the introduction of the segmentation masks in our input, so we pivoted to use a pretrained ResNet-18 model for Step 2. We chose ResNet-18 as opposed to ResNet-50 because we wanted to perform cross-validation on our model and thus wanted to optimize for a faster training time without too significant an accuracy loss.

We then performed cross-validation of the ResNet-18 model for Step 2. We tested ResNet-18 with every combination of regularization strengths of 3e-4, 5e-4, and 7e-4 and epoch numbers 3, 5, and 7. We found that a regularization rate of 5e-4 and 7 epochs was the most successful combination, so we trained on this combination. As in Step 1, we had to alter the final layer to output 50 class scores to accord with the number of scenes in our dataset. We used the Adam optimizer, chosen because of its superior performance across CNN tasks.

For Step 3 (Creating Segmentation Masks), we turn back to the ResNet-50 pretrained model to produce semantic segmentation masks for our raw images. For this step, we had to determine the number of object classes across the images by hand, since we took a sample of images from the full ADE20K dataset. We found that we had 253 classes, small enough to fit into a 1-color channel greyscale representation for our segmentation masks with values [0, 252]. This time, we do a similar procedure to Step 1 but with 253 output classes coded into the last layer of our model. We again use the Adam optimizer for the reasons detailed above.

Finally, for Step 4 (Scene Classification from Raw Images and Custom Segmentation Masks), we used our best ResNet-18 model determined during cross validation in Step 2 combined with our segmentation model from Step 3 to produce our custom raw image/segmentation mask concatenation input and our output of class scores for each image. We only evaulated the test set at this part, so we did not train at this step.

3.4. Loss Functions

For Steps 1, 2, and 3, we needed to find the loss as a function of the class scores for each image compared to the true class of the image. This setup – of comparing class labels to class scores – is well suited to the Softmax loss function (also called the cross-entropy loss, as in our model). Recall that Softmax loss is defined as:

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

Where L_i is the loss for each image and f_j is the j-th element of the vector of class scores f. In words, the softmax loss takes a vector of class scores and returns a same-length vector of scores between 0 and 1 that sum to 1, giving us ”probabilities” for each class. We decided to use Softmax rather than hinge loss or other loss functions because Softmax returns a vector of probabilities for each image that we can feed into the next training iteration rather than a single class score, as in hinge loss. Softmax is also differentiable and more resistant to outliers, making it a good choice for CNNs on many thousands of images.

For Step 3 (Creating Segmentation Masks), we want to create a segmentation mask output, giving an object class score vector to each pixel within an image. We can again use the Softmax loss function for this step, where instead of summing the class score losses across the whole image, we sum the object score losses across each pixel and return an array of probabilities for each pixel in each image. We chose to continue with Softmax (without reduction) for Step 3 because it is relatively simple, fast, and effective for semantic segmentation. We could have also used Dice loss here, but decided to stay with a simpler model.

4. Dataset and Features

Our inputs included the raw images, segmentation masks, and class labels from the ADE20K dataset. The original ADE20K dataset contains 25,574 training images and 2,000 test images across 749 scene classes. Some scene classes only had a few corresponding images, and the entire dataset also took up more storage space than we were able to store on our machines locally or process in RAM on Google Colab.

Thus, we decided to filter our input images to keep only the 50 most common classes. We wrote a script called `ade20k_scene_parser.py` to find the appropriate data to classify the these 50 scenes. The script generates a list of training, validation, and test indices to use based on the number of scenes we want to classify. It also creates an index-to-scene dictionary as well as a scene-to-index dictionary (for classification purposes) for the underlying images. After processing all of the original ADE20K data with our parsing program, we were left with 14,550 training images and 1,211 test images across the top 50 classes. We created a validation set from the training images by parsing images from all 50 scenes such that the validation set would be roughly 1/10th the size of the training set, leaving us with 1,369 validation images and 13,181 training images.

With these arrays and dictionaries assembled, we created another program called `load_ade20k_data.py` to check that our indices are non-overlapping and to load the correct indexed images into nine different arrays. The first three arrays are for the training, validation, and test raw images Each array is of shape `(N, 128, 128, 3)`, where `N` is the number of images of a given type (raw pixel training images, for example), 128 is the side length of the square image (we chose 128 for side lengths because of RAM constraints), and 3 is the number of RGB channels for a given image. The second three arrrays contain the training, validation, and testing segmentation masks corresponding to the same images in the raw image arrays. The segmentation mask arrays provided had shape `(N, 128, 128, 3)`, identical to the raw image arrays, except that we only want 1 color channel instead of 3 since the segmentation masks are simply classifying each pixel by its object class which is representable within a single [0, 255] pixel value. We represent the mask in a single channel using PIL's ImageOps package to greyscale each segmentation mask pixel to give us a shape of `(N, 128, 128, 1)`. Finally, our third set of three arrays contains the scene labels corresponding to the training, validation, and test raw images. All of this preprocessing gave us access to raw pixel images, segmentation mask labels, and scene class labels for training, validation, and testing that we were able to store and manipulate within the Colab environment without exhausting RAM.

5. Experiments, Results, and Discussion

5.1. Experiments

For each of our experimental steps, we used a batch size of 64 since CUDA could not handle a larger batch size given our models. We employed a learning rate of 1e-3, which, according to Mishra et al., is the optimal learning rate for ResNet models [4]. We also used the Adam optimizer across steps since Adam tends to show superior performance for CNN tasks. To summarize our individual experiments, we used the following algorithms/optimizations for each of our steps:

1. For Step 1 (Vanilla Scene Classification), we used ResNet-50 with an altered fully connected layer at the end to produce our 50 class scores for each image. To sanity check our model, we first overfit our model on

64 training examples using the Adam optimizer with zero weight decay. We were able to achieve 100% accuracy on the training examples after only 100 iterations. We then proceeded to use ResNet-50 and the Adam optimizer with a weight decay of 1e-4 to prevent overfitting.

2. For Steps 2 and 4 (Vanilla Scene Classification with Segmentation Masks), we started with our own 24-layer model that was able to perfectly overfit to 64 training examples after 100 iterations but achieved suboptimal performance after training. To improve accuracy, we instead used ResNet-18 with an altered fully connected layer at the end to produce our 50 class scores for each image. We used ResNet-18 rather than ResNet-50 in this step so we could cross-validate hyperparameters without exhausting Google Colab resources. Our cross-validation tuned a combination of weight decay levels and epoch numbers to achieve the best possible model (described in more detail in the Methods section). We ended up with a best model that used a L2 regularization rate of 5e-4 and 7 epochs.

3. For Step 3 (Creating Segmentation Masks), we used the ResNet-50 network with 253 class labels in the fully connected layer, matching the number of object classes in the dataset. We used a weight decay of 1e-5 and 8 epochs (we did 4 epochs, saved our model, and then performed another 4 epochs to stay with Google Colab's resource limits).

5.2. Results and Discussion

The following table details our high-level results:

Results		
Step	Final Val. Accuracy	Final Test Accuracy
Step 1	41.71%	**53.26%**
Step 2	49.23%	**60.03%**
Step 3	N/A	N/A
Step 4	–	**48.31%**

We present results for each of our four steps sequentially.

5.2.1 Step 1: Vanilla Scene Classification

Our baseline scene classification model using ResNet-50 was able to achieve **84.09%** accuracy on the training set, **41.71%** accuracy on the validation set, and **53.26%** accuracy on the test set. The model was thus moderately overfitting to the training data, though we were unable to create a more generalizable model without harming validation accuracy.

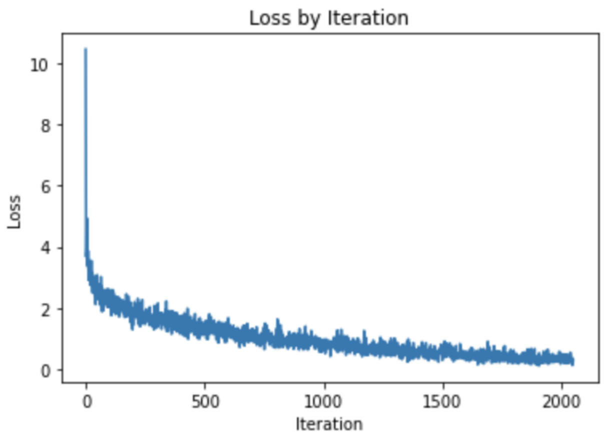

Figure 4. Plotted training loss from vanilla scene classification using ResNet-50 over 2,000 iterations.

Our loss on the training data can be seen in Figure 4. Our loss very quickly drops below 3, and then continues to decline with noise through our 2,000 iterations, the slope of decline getting shallower across iterations.

We find it a bit strange that our validation accuracy is significantly lower than our testing accuracy. This may be because of the way we selected images for our validation set, which is small and might not have the same proportion of images from each scene class as the training set and testing set. This continues to be an issue through the rest of our models, indicating that a different way of choosing the validation set might lead to improved validation accuracy.

5.2.2 Step 2: Scene Classification from Raw Images and Ground-Truth Segmentation Masks

When we concatenated the ground-truth segmentation masks from the ADE20K dataset with our raw images and ran a tuned ResNet-18 model on the data, we saw our training accuracy jump to **91.19%**, our validation accuracy jump up to **49.23%**, and our testing accuracy jump to **60.03%**. We again see the moderate overfitting from Step 1, but also see our validation and test accuracy increase by 7 to 8-percentage points between the two models.

This is theoretically promising: it shows that if the segmentation masks are learned accurately from the raw image data, then concatenating a 1-channel segmentation mask to the original raw image can significantly improve scene classification accuracy for complex scenes. Our result is especially impressive considering that we downgraded from ResNet-50 to ResNet-18 for this step, meaning that ResNet-50 applied to the concatenated input would likely have improved our accuracy even more. This is the result we hoped to show, and will attempt to show with our own custom segmentation masks.

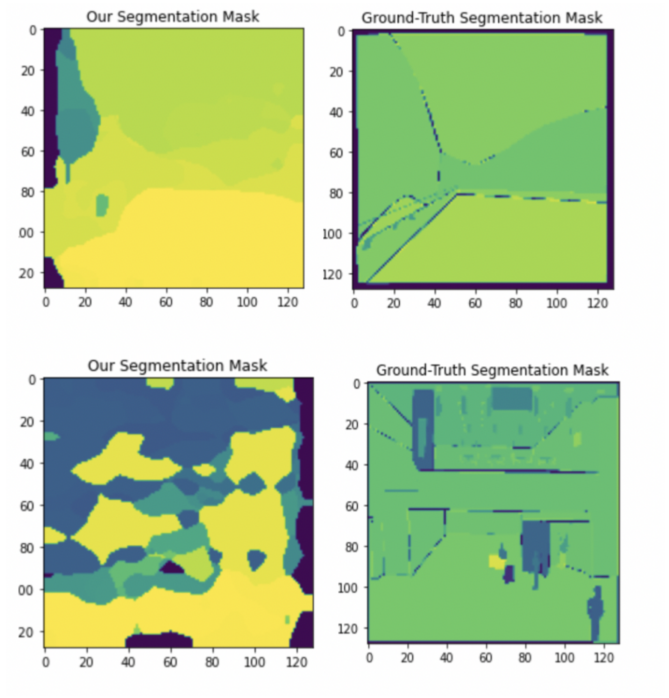

Figure 5. Examples of our segmentation masks (left) next to the corresponding ground-truth segmentation masks (right).

5.2.3 Step 3: Creating Segmentation Masks

For our custom segmentation masks, we saw our pixel-wise loss decrease over 8 epochs. Figure 5 shows some examples of our custom segmentation masks learned on our test set compared to the corresponding ground-truth segmentation masks. We see that for the first example, the custom segmentation mask appears to have similar shapes to the ground-truth segmentation mask but with different coloring and some random splotches that do not correspond to the ground-truth mask. For the second example, we see that our custom segmentation mask is very splotchy and does not seem to correspond to the ground-truth segmentation mask.

Unfortunately, we were not able to improve our segmentation mask creation after trying a number of custom CNN models. ResNet-50 seems to perform well on simpler segmentation masks but struggles when there are complex and small shapes within an image. While some of our segmentation masks appear to recognize the spirit of an image, we are not confident that our learned segmentation masks will improve classification accuracy when concatenated with the raw images in Step 4.

One possible reason for our failure to produce high-fidelity segmentation masks comes from our data preprocessing step, where we condensed each ground-truth segmentation mask to only 1 color channel using the ImageOps greyscale function described in the Methods section. We are unsure if converting each segmentation mask to greyscale made it difficult or impossible for our own ResNet learning algorithm to learn the masks, given that the masks may no longer represent the raw images once greyscaled. We did not attempt to use the unprocessed, 3-channel segmentation masks because we could not learn the masks using simple object classification for our 253 object classes on the pixels if the outputs were 3 channels per pixel. Potentially finding a dataset with 1-channel segmentation masks or finding a way to keep 3-channel segmentation masks for training would have improved our results.

The segmentation masks may have also been suboptimal for training regardless of greyscale issues. After looking at a large number of the ground-truth segmentation masks beside the raw images they represent, we are not convinced that the ADE20K dataset offers the best possible segmentation masks for training. Many of the segmentation masks appear to be missing features from the raw images or arbitrarily selecting which features to include as objects and which to ignore. We believe our model might create better segmentation masks with an underlying dataset composed of segmentation masks that better mirror the raw images they represent.

5.2.4 Step 4: Scene Classification from Raw Images and Custom Segmentation Masks

As expected, our ResNet-18 model using concatenated raw images and segmentation masks performs worse when using our custom segmentation masks than when using the ground-truth masks in Step 2. We find that using our custom segmentation masks reduces the testing accuracy to **48.31%**, a 12-percentage point decrease from the ground-truth segmentation mask attempt. It also performs slightly worse than using the raw images alone, meaning that we would be better off not using our segmentation masks altogether and simply performing vanilla scene classification from raw images.

Given how low our pixel-wise accuracy is in Step 3 for creating segmentation masks, we are unsurprised that using our custom segmentation masks concatenated with the raw images performs poorly. However, we demonstrated our approach using segmentation masks concatenated with the raw images can improve scene classification accuracy if those segmentation masks are highly accurate, as shown in Step 2.

6. Conclusion and Future Work

Our project attempts to improve the accuracy of scene classification using the ResNet class of models by concatenating the raw scene image with a learned segmentation mask of that image. In theory, concatenating the segmentation masks to our raw images gives the final neural network more information about the objects and their placement within a scene, which may provide better predictive power than the raw pixels alone through object patterns among scenes.

We see that concatenating the ground-truth segmentation masks for our images with the raw images improves our model's scene classification accuracy by nearly 7 percentage points, a jump made even more salient by the fact that our combined input model used only 18 layers instead of the 50 for vanilla scene classification. This proves that our approach has validity: with well generated segmentation masks, we can improve scene classification accuracy. We then attempt to learn our own segmentation masks from the ground-truth ones provided in the ADE20K dataset, we see very mixed results from our segmentation mask creation model. When we combine our custom segmentation masks with the raw images, we see slightly decreased accuracy from our vanilla classification model, indicating that our segmentation masks were poorly learned. We suspect that the ground-truth segmentation masks from the ADE20K dataset are not perfectly designed from the raw images after viewing many images alongside their segmentation masks.

Future work to extend this paper would largely involve better learning of the segmentation masks to allow future models to outperform vanilla scene classification without using ground-truth segmentation masks. We suspect that using a larger and more carefully designed dataset could aid in this effort, as well as tweaking the contents of ResNet-50 more precisely to improve pixel-wise accuracy. It is possible that a different pixel-wise loss function, such as Dice loss, may improve our results.

7. Contributions

Matthew Kaplan produced the scripts needed to pull our training, validation, and testing data from the ADE20K files. He also wrote the vast majority of the model code and annotated the final notebook with insights about our procedure and results. He researched the use of ResNet for our project and performed the iterative testing and cross-validation of hyperparameters.

Avery Rogers worked on the theoretical aspect of creating segmentation masks by researching the proper loss function for semantic segmentation. She also wrote the vast majority of the final paper, including conducting the literature review and compiling all figures, tables, and mathematical equations/theoretical explanations. She also created the slides for the final presentation.

Both partners are satisfied with the contributions of the other to the project as a whole. The work was fairly divided and they are happy with the product.

8. Supplementary Material

Code for this project is organized in the following GitHub repository. URL if hyperlink does not work: https://github.com/mkaplan1-Stanford/CS231n-Project

References

[1] N.W. Campbell, W. Mackeown, B.T Thomas, and T. Troscianko. The automatic classification of outdoor images, 1997. 2

[2] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015. 1, 3

[3] L.J. Li, H. Su, Y. Lim, and L. Fei-Fei. Objects as attributes for scene classification, 2012. 2

[4] S. Mishra, T. Yamasaki, and H. Imaizumi. Improving image classifiers for small datasets by learning rate adaptations, 2019. 4

[5] J. Yao, S. Fidler, and R. Urtasun. Describing the scene as a whole: Joint object detection, scene classification and semantic segmentation, 2012. 2

[6] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. Scene parsing through ADE20K Dataset, 2017. 1