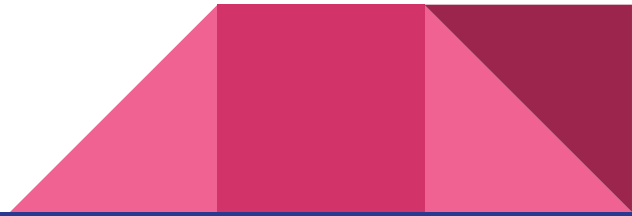
Urban Area Detection System

with Transfer Learning

Bo-Yan Huang
2024 1/16

Introduction - Background

Contemporary vehicles are equipped with built-in cameras that enable them to perceive their surroundings, and some even possess self-driving capabilities. A car endowed with urban area detection features utilizes cutting-edge sensor technologies and machine learning algorithms. This allows it to smartly identify and react to urban environments, enhancing safety and providing a driving experience finely tuned to the intricacies of city living.



Introduction - Challenges

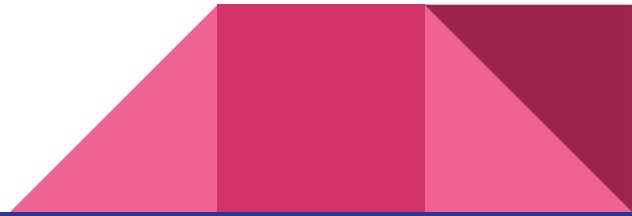
- **Data Quality and Variability:** Developing robust machine learning algorithms requires diverse and high-quality training datasets that accurately represent the complexity of urban environments. Obtaining and curating such datasets can be resource-intensive.
- **Model Generalization:** Urban landscapes can be highly diverse, encompassing different architectural styles, road layouts, and infrastructure. Adapting a system to recognize this variability poses a significant challenge.
- **Real-time Inference:** Real-time classification of human activities while managing power consumption on a car device is a critical challenge. The app needs to efficiently process sensor data, make predictions promptly, and operate within the constraints of limited battery resources.

Transfer learning

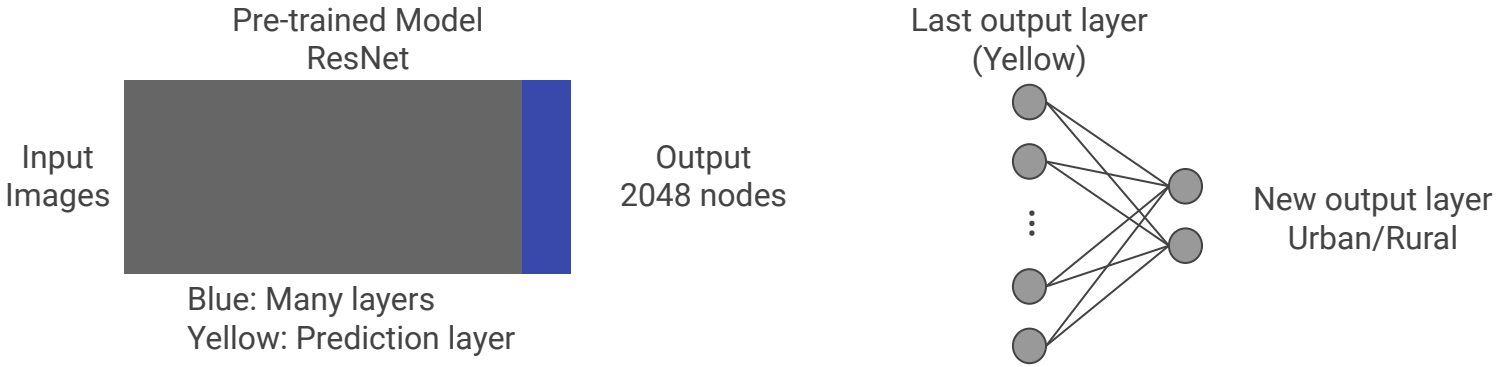
Transfer learning addresses several challenges associated with building machine learning models, especially when dealing with limited data or specific use cases.

Pre-trained models, which have learned from vast and varied datasets, can capture generic features, patterns, and representations. This knowledge can be transferred and fine-tuned for the specific nuances of urban or rural environments, reducing the need for an exhaustive dataset.

Transfer learning allows leveraging the knowledge acquired by a model on a different but related task. This significantly reduces the need for extensive labeled data, making it possible to achieve good results even with limited data for the new use case.

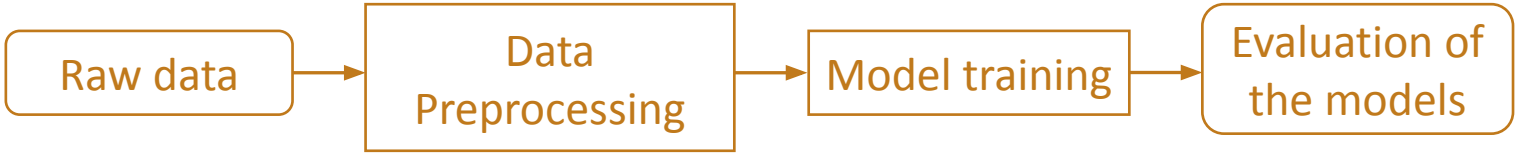


Transfer learning



We're replacing the final layer of the ResNet model with a new dense layer featuring two nodes. One node assesses urban characteristics, and the other evaluates rural qualities. All nodes in the layer before prediction contribute to gauging urban attributes, and connections illustrate these potential relationships. Similarly, the information at each node influences our assessment of the photo's rural characteristics. Finally, we employ softmax to obtain the predicted class.

Flowchart of urban area detection system



A collection of images sized 225x225 pixels with a color depth of 24 bits.

Resizing images
Converting color depth to RGB 256
Rescale the pixel values
Data augmentation

Select a suitable pre-trained model and fine-tune the additional layers to create a new model.

Assess the model to determine if it meets our requirements. If not, revisit the previous step for further refinement.

About data

This dataset consists of images sized 225x225 pixels with a color depth of 24 bits. It was compiled from a Google Image search, encompassing both rural and urban subjects. The search was limited to images in the public domain, specifically those with a 1:1 aspect ratio. This dataset proves valuable for experimenting with diverse image processing techniques.

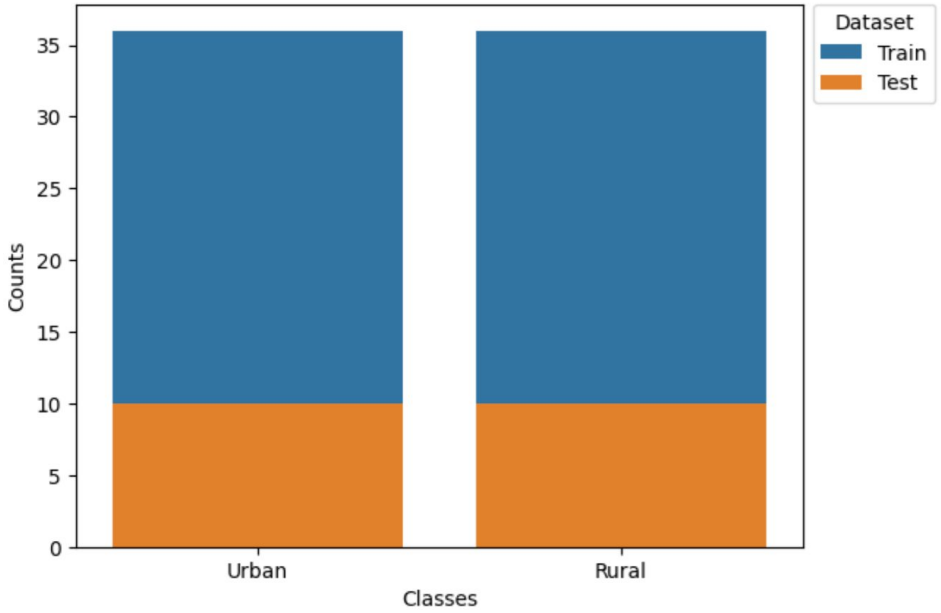
Rural




Urban




Label counts per category

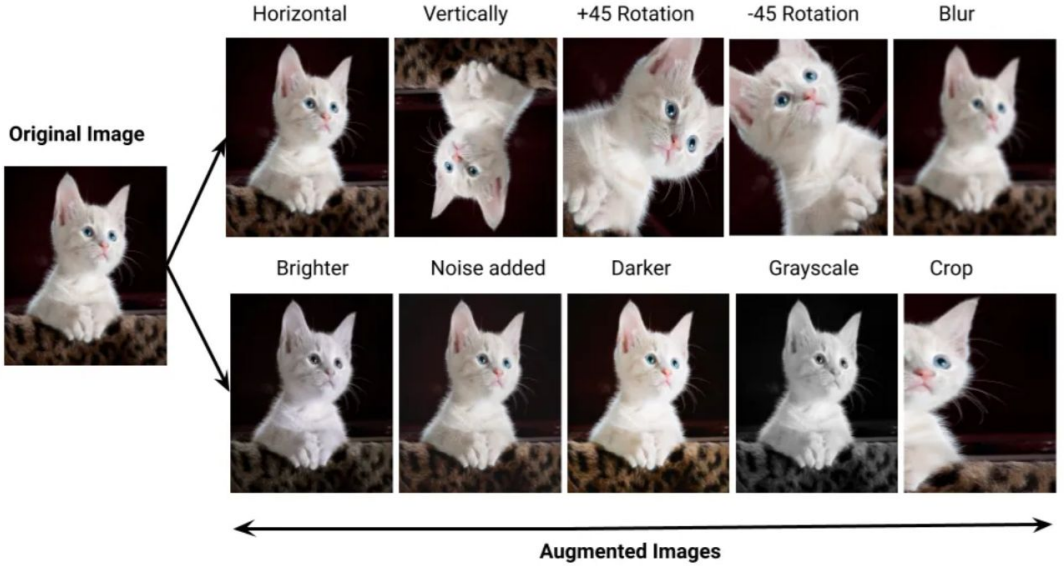


The dataset is divided into training data, comprising 36 images for each urban and rural scene, and a test set, which incorporates 10 images for each category. The labeling is based on the directory names of the respective image files.

Data augmentation

Data augmentation is a low-cost and effective approach to improving the performance and accuracy of machine learning models in data-constrained scenarios.

In our specific scenario, unrestricted rotation is not feasible due to the necessity for the building to maintain an upright appearance. Nevertheless, horizontal flipping and cropping remains a viable option.



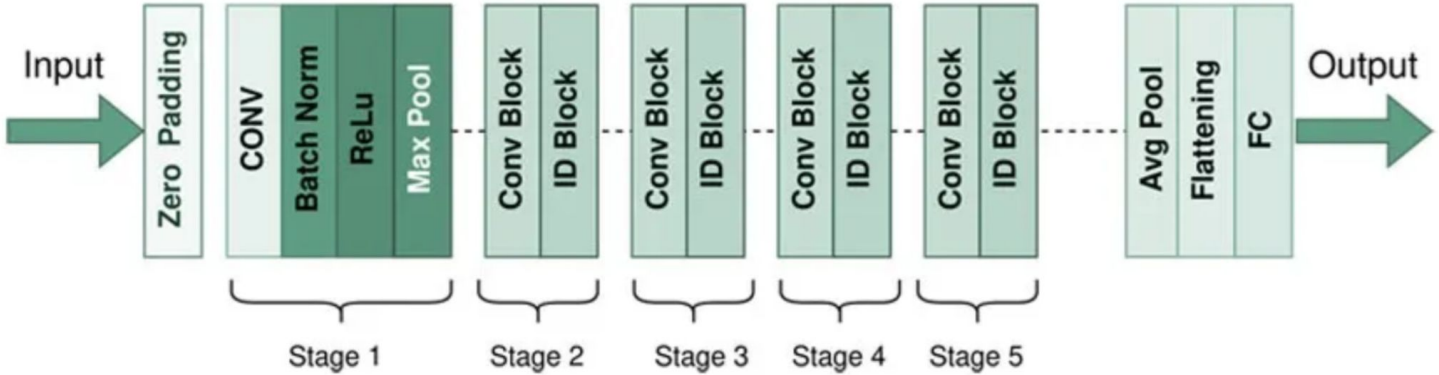
Pre-trained Models

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9

Keras Applications offer pre-trained deep learning models with accompanying weights. These models serve various purposes, including prediction, feature extraction, and fine-tuning.

The the table to the left displays some of the available pre-trained models. For the complete list, refer to https://keras.io/api/applications/.

ResNet50

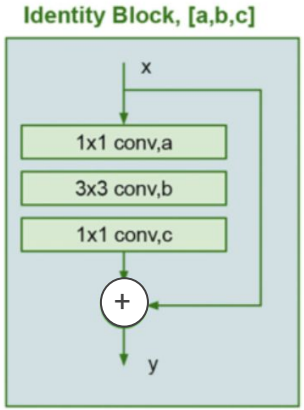


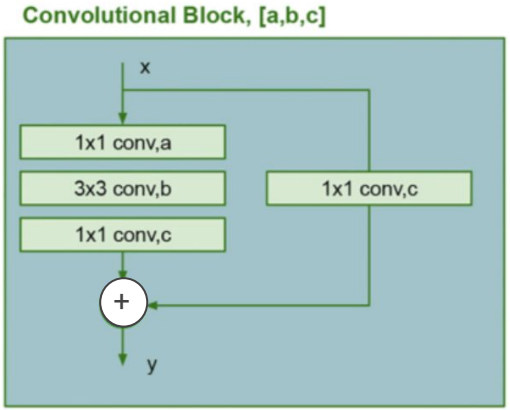
The Residual networks (ResNet in short) architecture is one of the most well known and popular networks in the literature, introduced by Microsoft Reseach in 2015 in the paper written by He. et. al. The residual networks proposed in this paper won the first places on ILSVRC 2015 classification task, ImageNet detection, ImageNet localization, COCO detection and COCO segmentation task.

ResNet50

The identity block is a residual block that preserves the input information by bypassing the identity of the input directly to the output. It consists of three main layers: two convolutional layers with smaller filter sizes and a batch normalization layer. These layers operate in a sequence without any skip connections. The skip connection helps mitigate the vanishing gradient problem, facilitating the flow of gradients during backpropagation.

The convolutional block, also known as a residual block with a convolutional shortcut, includes a convolutional layer in the shortcut connection. It involves three layers: a convolutional layer with a larger filter size, a batch normalization layer, and a rectified linear unit (ReLU) activation function. The convolutional shortcut provides an alternative route for gradient flow, enhancing the ability of the network to learn complex features.





ResNet50

We categorize photos into two classes: urban and rural, denoted as num_classes. Moving on to building the model, we configure a sequential model to which we can add layers. Initially, we incorporate a pre-trained ResNet model. By setting include_top=False during the ResNet model creation, we indicate the exclusion of its last prediction-making layer. Additionally, we use a file without the weights for that layer.

The argument pooling='avg' specifies that, if there are extra channels in the tensor at the end of this step, we want to condense them into a 1D tensor by taking an average. This results in a pre-trained model that generates the layer depicted in the graphic. Subsequently, we add a Dense layer for predictions, specifying the number of nodes in alignment with the number of classes. Finally, we apply the softmax function to generate probabilities.

```
num_classes = 2
resnet_weights_path = '../input/resnet50/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5'

my_new_model = Sequential()
my_new_model.add(ResNet50(include_top=False, pooling='avg', weights=resnet_weights_path))
my_new_model.add(Dense(num_classes, activation='softmax'))

# Say not to train first layer (ResNet) model. It is already trained
my_new_model.layers[0].trainable = False
```

ResNet50

The parameters from ResNet are frozen, allowing only the last layer to be trainable. Therefore, the total number of trainable parameters is equal to twice the number of nodes in the last layer (2 * 2048), resulting in 4096 parameters.

We employ stochastic gradient descent (SGD) to minimize the categorical cross-entropy loss. Additionally, we instruct the code to provide the accuracy metric, representing the fraction of correct predictions. This metric is more straightforward to interpret compared to categorical cross-entropy scores, making it beneficial to print and assess the model's performance.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Functional)        (None, 2048)              23587712
_________________________________________________________________
dense (Dense)                (None, 2)                 4098
=================================================================
Total params: 23,591,810
Trainable params: 4,098
Non-trainable params: 23,587,712
_________________________________________________________________
```

ResNet50 Results

In [5]:

```
my_new_model.fit(
        train_generator,
        steps_per_epoch=6,
        validation_data=validation_generator,
        validation_steps=1)
```

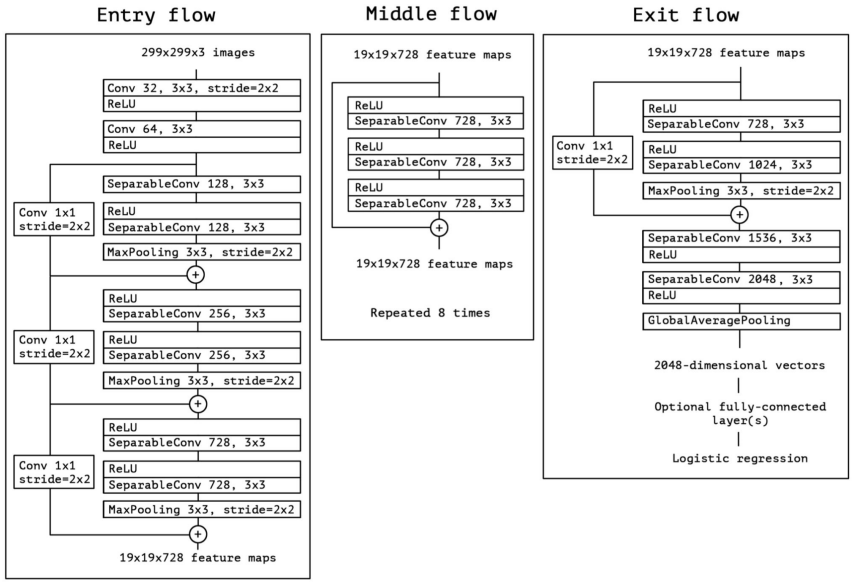
Without data augmentation:

```
6/6 [==============================] - 6s 646ms/step - loss: 0.5250 - accuracy: 0.7639 - val_loss: 0.3172 - val_accuracy: 0.900
0
```

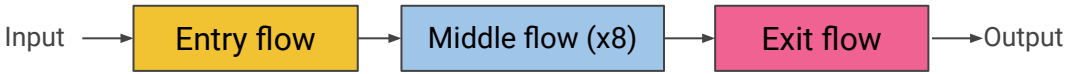
With data augmentation:

```
6/6 [==============================] - 6s 655ms/step - loss: 0.3363 - accuracy: 0.8611 - val_loss: 0.1408 - val_accuracy: 1.000
0
```

Xception



Let's delve into another pre-trained model to evaluate the efficacy of transfer learning. Xception, which was emphasized in the previous slide as the most accurate model, achieves a top-1 accuracy of 79% and a top-5 accuracy of 94.5%. Derived from InceptionV3, the Xception model is named for its adapted method, Extreme Inception.
In Xception Model, the data frist goes through the entry flow, then through the middle flow which is repeated eight times, and finally through the exit flow. The architecture is shown to the left.



Xception

The total number of trainable parameters is equal to twice the number of nodes in the last layer (2 * 2048), resulting in 4096 parameters.

Similar to previous the previous configuration, we utilize stochastic gradient descent (SGD) to minimize categorical cross-entropy loss. We also specify the accuracy metric in the code, offering a more easily interpretable measure of the model's performance in terms of correct predictions, as opposed to categorical cross-entropy scores.

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 xception (Functional)       (None, 2048)              20861480

 dense_8 (Dense)             (None, 2)                 4098

=================================================================
Total params: 20865578 (79.60 MB)
Trainable params: 4098 (16.01 KB)
Non-trainable params: 20861480 (79.58 MB)
_________________________________________________________________
```

Xception Results

In [5]:

```
my_new_model.fit(
        train_generator,
        steps_per_epoch=6,
        validation_data=validation_generator,
        validation_steps=1)
```

With data augmentation:

```
6/6 [==============================] - 5s 620ms/step - loss: 0.5840 - accuracy: 0.7083 - val_loss: 0.5371 - val_accuracy: 0.850
0
```

Compare the performance of models

Pre-trained Model	Accuracy on Testing dataset	Accuracy on Training dataset	Training time
ResNet50	100%	90%	6s
Xception	85%	71%	5s

Conclusions

- The effectiveness of transfer learning becomes apparent when employed for a new task. The resulting model predicts whether an image belongs to a rural or urban category with satisfactory accuracy.
- Transfer learning allows us to train only the additional layers, significantly reducing both training time and the computational resources required.
- Even a small dataset can suffice for training a satisfactory model.
- Data augmentation enhances the performance and robustness of machine learning models by artificially expanding the dataset and exposing the model to a diverse range of input variations.

Outlook

Possible issues:

- **Limited Representations:** The pre-trained model may not have encountered sufficient diversity in the small dataset (ex. the weather are basically the same in the rural dataset), limiting its ability to represent the full range of variations in the new task.
- **Fine-tuning Challenges:** With a small dataset, fine-tuning the pre-trained model's parameters may be challenging. The model might not converge effectively, or it may be sensitive to hyperparameter choices.
- **Transferability of Features:** The features learned on a large dataset may not be as transferable to a small, task-specific dataset. The relevance of pre-trained features may be diminished if the new task requires unique or specific representations.
- **Domain Shift:** If there is a significant domain shift between the pre-training dataset and the new task dataset, the transferred knowledge may not align well with the target task.

Outlook

Possible solutions:

- **Limited Representations:** We need to correct more images encompassing various weather conditions, day and night settings, and different seasons.
- **Fine-tuning Challenges:** Expanding the dataset using data augmentation techniques can alleviate problems associated with fine-tuning.
- **Transferability of Features:** Explore alternative pre-trained models that have been trained on datasets similar to the one used in our task. Additionally, consider models that have been trained for comparable tasks.
- **Domain Shift:** Regularly evaluate the model's performance on the target domain during training. Monitor domain shift indicators and adjust models accordingly.

Appendix

Data source: https://www.kaggle.com/datasets/dansbecker/urban-and-rural-photos

Pre-trained models: https://keras.io/api/applications/

Courses: https://www.coursera.org/learn/deep-learning-reinforcement-learning/home

Jupyter notebook: https://github.com/r95222023/IBM-Machine-Learning-Professional-Certificate/tree/main/Deep%20Learning%20and%20Reinforcement%20Learning