



APPLIED MACHINE LEARNING - COS30082

LAB 06 - TRANSFER LEARNING

Instructor:
Mr. Minh HOANG

Student:
Huynh Trung Chien
104848770



SEPTEMBER 2025

COS30082 - Applied Machine Learning

Week 6 - Lab - Transfer Learning

1. Explain the concept of transfer learning. How does it differ from training a model from scratch?

a. Training from scratch

Training a model from scratch requires us to prepare everything for the model, including:

- **Large Dataset** - Building a model from scratch requires a very large dataset to learn features from simple to complex. When we do not have enough data, the model might not perform well; it might cause overfitting or underfitting.
- **High Computational Power** - It also requires very powerful hardware because the weights of the models are randomly initialized and updated through intensive training.
- **Long Training Time** - For large model with many parameters, training can take a huge amount of time. The more complex the model is, the more time it needs to converge.
- **Huge Hyperparameter Tuning** - To make the model to perform as expected, we need to experiment with learning rates, optimizers and other methods. It could take intensive amount of time for making modification every time.
- **Task-specific Feature Learning** - When we want the model to perform another task, we have to train the model again from scratch with the specific feature for that task.

b. Transfer Learning

In terms of transfer learning, we do not need to prepare everything because we rely on the existing models, which were trained on a large dataset. Therefore, the tasks when doing transfer learning is less heavy:

- **Smaller Dataset** - Because we use a large-dataset-pretrained model, the model can still perform well if we have a limited amount of data.
- **Moderate Computational Power** - When we use transfer learning, we only need to fine-tune some layers of the models (not all of them). Therefore, training is much faster and less computationally intensive.
- **Shorter Training Time** - The training time will also be faster.

- **Less Hyperparameter Tuning** - Because the hyperparameters of the model have been tuned before, we only need to tune some of the parameters.
- **Feature Reuse** - Finally, the general features from the pretrained model can be reused for other tasks.

2. What is fine-tuning in the context of transfer learning, and why is it useful?

Fine-tuning is the process of unfreezing some of the layers of the pretrained model and use those layers to train on the new dataset, with a low learning rate. Fine-tuning is mainly used when we want to use the model for a specific task, then we utilize the general features from the pretrained model and adapt them to the specific patterns for the new task.

Below are the steps that we usually perform when doing fine-tuning:

- Choosing a pretrained model that has already learned general features from a large dataset.
- Changing the classification layers, normally changing the output layer (the number of output classes).
- Freezing the base layers, and training on the top layers (feature extraction).
- Unfreezing some of the deeper layers and train the network from the first unfreezed layer onwards with a very small learning rate (fine-tuning).

Fine-tuning is useful because:

- **Better Generalization and Domain Adaptation**- We can adapt the model with a new dataset for a new task with higher accuracy compared to training a model from scratch. This is because we take advantage of the general features built for other tasks.
- **Efficient Use of Resources** - We can reduce the training time and computation because we reuse the existing features and fine-tune some of the layers.
- **Effective with Small Datasets** - We can get a very good performance without the need to prepare a large dataset.

3. Why is it important to freeze the convolutional base during feature extraction?

When doing feature extraction, it is crucial to freeze the convolutional base because:

- **Preserving Learned Features** - The convolutional base has learned the general features from a very large dataset. Therefore, if we freeze it, the general features will not be changed by the new data.
- **Preventing Overfitting** - When the dataset is small, training all layers can cause the model to memorize the data and cause overfitting.

- **Reducing Computation** - When we freeze the convolutional base, we will train with fewer parameters, making the process less GPU-intensive.
- **Efficient Learning** - By freezing the convolutional base, the models will focus on learning the complex features of the task rather than learning the low-level features.

4. Why use data augmentation?

Data augmentation is a technique to expand and diversify the training dataset, some common transformations that we usually use for augmentation include:

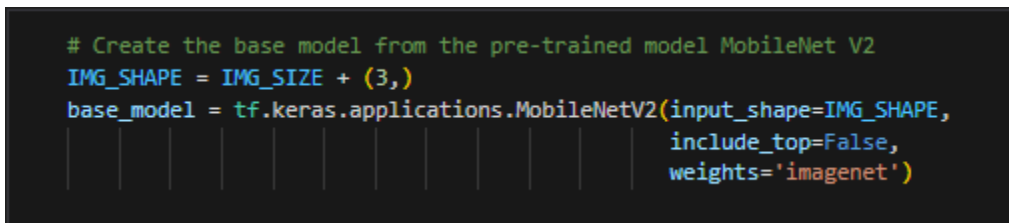
- Flipping (horizontal or vertical)
- Rotation
- Zooming
- Cropping
- Translation
- Brightness
- Adding noise

Data augmentation is useful because:

- **Preventing Overfitting** - When we do not have a large dataset, we can use data augmentation to create multiple variations of the same pictures, making the dataset more diverse, thus reducing overfitting.
- **Improving Robustness** - This method helps the model to deal with real-world variations where lighting, rotation, noise, and flipping usually happen.
- **Encouraging Invariance** - Some transformations, such as cropping and zooming, help the model to focus on key object rather than not important information.
- **Increasing Performance** - Data augmentation is a good method to improve the overall performance of the model.

5. Take a screenshot of the code snippet where the pre-trained MobileNetV2 model is loaded without the top classification layers.

Figure 1 shows the code for loading the MobileNetV2 model without the top classification layers (include_top=False).



```
# Create the base model from the pre-trained model MobileNet V2
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                                include_top=False,
                                                weights='imagenet')
```

Figure 1: Loading the MobileNetV2 model without top classification layer

6. Take a screenshot of the portion of code where the pre-trained model is set to be non-trainable for feature extraction purposes.

```
# we have to freeze the convolutional base when doing feature extraction
base_model.trainable = False
```

Figure 2: The base_model is set to non-trainable for feature extraction

7. Take a screenshot of the data augmentation layers defined in the model.

```
# Define the data_augmentation layer
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip('horizontal'),
    tf.keras.layers.RandomRotation(0.2),
    tf.keras.layers.RandomZoom(0.2),
])
```

Figure 3: Define the data augmentation layer

```
# Build a model
inputs = tf.keras.Input(shape=(160, 160, 3))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

Figure 4: The data augmentation layer is added to the model

8. Take a screenshot of the code that shows the addition of the new classifier layers on top of the base model.

```
# Add one global average pooling 2D
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)

(32, 1280)

# Apply a tf.keras.layers.Dense
prediction_layer = tf.keras.layers.Dense(1, activation='sigmoid')
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)

(32, 1)

# Build a model
inputs = tf.keras.Input(shape=(160, 160, 3))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

Figure 5: Global Average layer, Dense layer are added on top of the base model before fine-tuning.

9. Additional Works for this lab

In order to get more familiar with transfer learning, feature extraction and fine-tuning. I fine-tuned some other models for the CIFAR-10 problem in the last lab. In the “104848770_HuynhTrungChien_TransferLearning_Cifar10.ipynb” file, I performed the data preprocessing, train/test split and feature extractions for seven models (*VGG16*, *VGG19*, *ResNet50*, *ResNet101*, *InceptionV3*, *MobileNetV2*, *EfficientNet B0*) and observed their performance. The results of those models are shown below:

	Model	Best Epoch	Train Accuracy (Best)	Val Accuracy (Best)	\
6	EfficientNetB0	16	0.9888	0.9646	
3	ResNet101	15	0.9978	0.9511	
2	ResNet50	17	0.9985	0.9435	
0	VGG16	20	0.9979	0.9405	
1	VGG19	20	0.9973	0.9398	
4	InceptionV3	9	0.6851	0.6559	
5	MobileNetV2	18	0.7018	0.5593	
	Val Loss (Best)				
6			0.1369		
3			0.2287		
2			0.2358		
0			0.3219		
1			0.3261		
4			0.9765		
5			1.2633		

Figure 6: Transfer learning results on CIFAR-10

From the results, I have not done well with the InceptionV3 and MobileNetV2 on the CIFAR-10 dataset. Previously, I have done a single notebook using MobileNetV2 for CIFAR-10, and the result was around 93 percent. However, this time, I have tried several ways but the result was not optimistic. I'll try to find out the problem and fix it in the future. Additionally, it is a pleasure for me if I can get feedback from you about this issue.