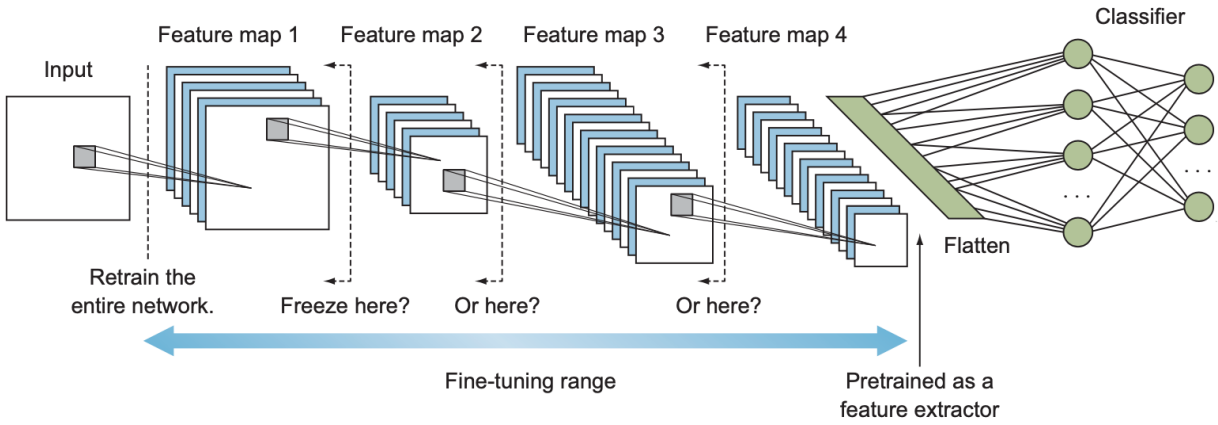
Transfer Learning for Image Classification: Part 2

There are two ways to use a pretrained network: feature extractor and fine-tuning. We now cover the fine-tuning.

As we discussed earlier, feature maps that are extracted early in the network are generic. The feature maps get progressively more specific as we go deeper in the network. Based on the domain similarity between the pretrained network and your classification task, we can decide to freeze the network at the appropriate level of feature maps:

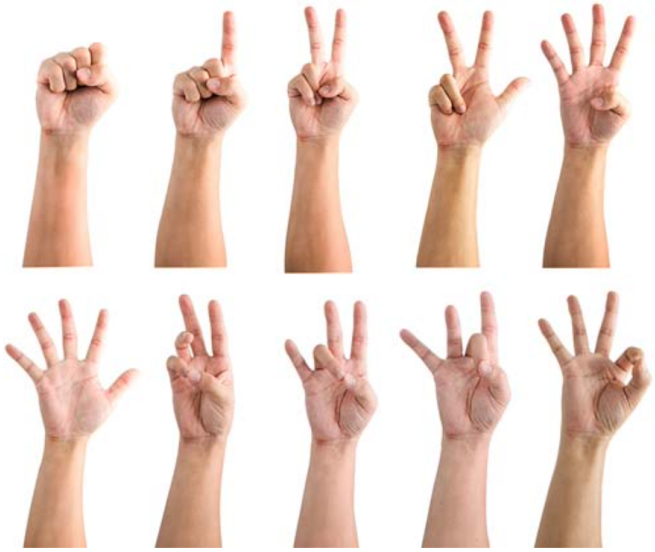
- If the domains are similar, we might want to freeze the network up to the last feature map level (feature maps 4, in the example).
- If the domains are very different, we might decide to freeze the pretrained network after feature maps 1 and retrain all the remaining layers.

Between these two possibilities are a range of fine-tuning options that we can apply. We can retrain the entire network, or freeze the pretrained network at any level of feature maps 1, 2, 3, or 4 and retrain the remainder of the network. We typically decide the appropriate level of fine-tuning by trial and error.



Project: fine tuning

In this project, we are going to explore fine tuning scenario where the target dataset is small and very different from the source dataset. The goal of this project is to build a sign language classifier that distinguishes 10 classes: the sign language digits from 0 to 9. Figure below shows a sample of our dataset.



Following are the details of our dataset:

- Number of classes = 10 (digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9)
- Image size = 100 × 100
- Color space = RGB
- 1,712 images in the training set
- 300 images in the validation set
- 50 images in the test set

It is very noticeable how small our dataset is. If you try to train a network from scratch on this very small dataset, you will not achieve good results. On the other hand, we were able to achieve an accuracy higher than 98% by using transfer learning, even though the source and target domains were very different.

For ease of comparison with the previous project, we will use the VGG16 network trained on the ImageNet dataset. The process to fine-tune a pretrained network is as follows:

1. Import the necessary libraries.
2. Preprocess the data to make it ready for the neural network.
3. Load in pretrained weights from the VGG16 network trained on a large dataset (ImageNet).
4. Freeze **part** of the feature extractor part.
5. Add the new classifier layers.
6. Compile the network, and run the training process to optimize the model for the smaller dataset.
7. Evaluate the model.

1. Import the necessary libraries

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications import imagenet_utils
from tensorflow.keras.applications import vgg16
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.metrics import categorical_crossentropy

from tensorflow.keras.layers import Dense, Flatten, Dropout,
BatchNormalization
from tensorflow.keras.models import Model

from sklearn.metrics import confusion_matrix
import itertools
import matplotlib.pyplot as plt
%matplotlib inline
```

2. Preprocess the data to make it ready for the neural network

Preprocess the data to make it ready for the neural network. Similar to the previous project, we use the `ImageDataGenerator` class from Keras and the `flow_from_directory()` method to preprocess our data. The data is already structured for you to directly create your tensors:

```
train_path  = 'dataset/train'
valid_path  = 'dataset/valid'
test_path  = 'dataset/test'

train_batches = ImageDataGenerator().flow_from_directory(train_path,
target_size=(224,224), batch_size=10)

valid_batches = ImageDataGenerator().flow_from_directory(valid_path,
target_size=(224,224), batch_size=30)

test_batches = ImageDataGenerator().flow_from_directory(test_path,
target_size=(224,224), batch_size=50, shuffle=False)
```

3. Load in pretrained weights from the VGG16 network trained on a large dataset (ImageNet)

Load in pretrained weights from the VGG16 network trained on a large dataset (ImageNet). We download the VGG16 architecture from the Keras library with ImageNet weights. Note that we use the parameter `pooling='avg'` here: this basically means global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor. We use this as an alternative to the Flatten layer before adding the fully connected layers:

```
base_model = vgg16.VGG16(weights = "imagenet", include_top=False,
input_shape = (224,224, 3), pooling='avg')
base_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
global_average_pooling2d (Gl (None, 512)               0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

4. Freeze **part** of the feature extractor part

Freeze some of the feature extractor part, and fine-tune the rest on our new training data. The level of fine-tuning is usually determined by trial and error. VGG16 has 13 convolutional layers: you can freeze them all or freeze a few of them, depending on how similar your data is to the source data. In the sign language case, the new domain is very different from our domain, so we will start with fine-tuning only the last five layers; if we don't get satisfying results, we can fine-tune more. It turns out that after we trained the new model, we got 98% accuracy, so this was a good level of fine-tuning. But in other cases, if you find that your network doesn't converge, try fine-tuning more layers.

```
# iterate through its layers and lock them to make them not trainable
with this code
for layer in base_model.layers[:-5]:
    layer.trainable = False

base_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
global_average_pooling2d (Gl (None, 512)               0
=================================================================
Total params: 14,714,688
Trainable params: 7,079,424
Non-trainable params: 7,635,264
```

5. Add the new classifier layers and build the new model

```
# use "get_layer" method to save the last layer of the network
last_layer = base_model.get_layer('global_average_pooling2d')

# save the output of the last layer to be the input of the next layer
last_output = last_layer.output
```

```
# add our new softmax layer with 3 hidden units
x = Dense(10, activation='softmax', name='softmax')(last_output)

# instantiate a new_model using keras's Model class
new_model = Model(inputs=base_model.input, outputs=x)

# print the new_model summary
new_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
global_average_pooling2d (Gl (None, 512)               0
_________________________________________________________________
softmax (Dense)              (None, 10)                5130
=================================================================
Total params: 14,719,818
Trainable params: 7,084,554
Non-trainable params: 7,635,264
```

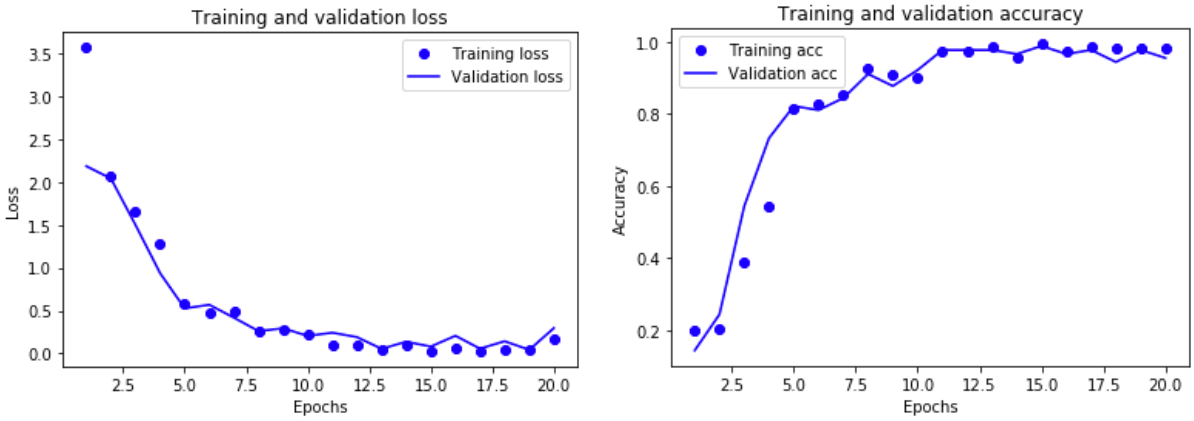
6. Compile the network, and run the training process to optimize the model for the smaller dataset

```
new_model.compile(Adam(lr=0.0001), loss='categorical_crossentropy',
metrics=['accuracy'])
from tensorflow.keras.callbacks import ModelCheckpoint

checkpointer = ModelCheckpoint(filepath='signlanguage.model.hdf5',
save_best_only=True)

history = new_model.fit(train_batches, steps_per_epoch=18,
          validation_data=valid_batches, validation_steps=3, epochs=20,
          verbose=1, callbacks=[checkpointer])
```

The model can be trained very quickly using regular CPU computing power.

7. Evaluate the model

We can plot the learning curves and perform evaluation from test set similar to the previous project



The testing accuracy can reach 98%. You can download the package `sign_language_project.zip` from the course website to reproduce and play around the results.