Transfer learning

Keras & TensorFlow

Transfer learning

Transfer learning (TL) is a research problem in ML that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. **For example**, knowledge gained while learning to recognize Cats could apply when trying to recognize Tigers.

Building Complex Models - Functional API

```
input_   = keras.layers.Input(shape=X_train.shape[1:])
hidden1 = keras.layers.Dense(30, activation="relu")(input_)
hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)
concat   = keras.layers.Concatenate()([input_, hidden2])
output    = keras.layers.Dense(1)(concat)
model    = keras.Model(inputs=[input_], outputs=[output])
```

Once you have built the Keras model, everything is exactly like earlier, so there's no need to repeat it here: you must compile the model, train it, evaluate it, and use it to make predictions.

- model.compile(loss="mse", optimizer="sgd")

- history = model.fit(**X_train**, **y_train**, epochs=20,
- validation_data = (**X_valid**, **y_valid**))

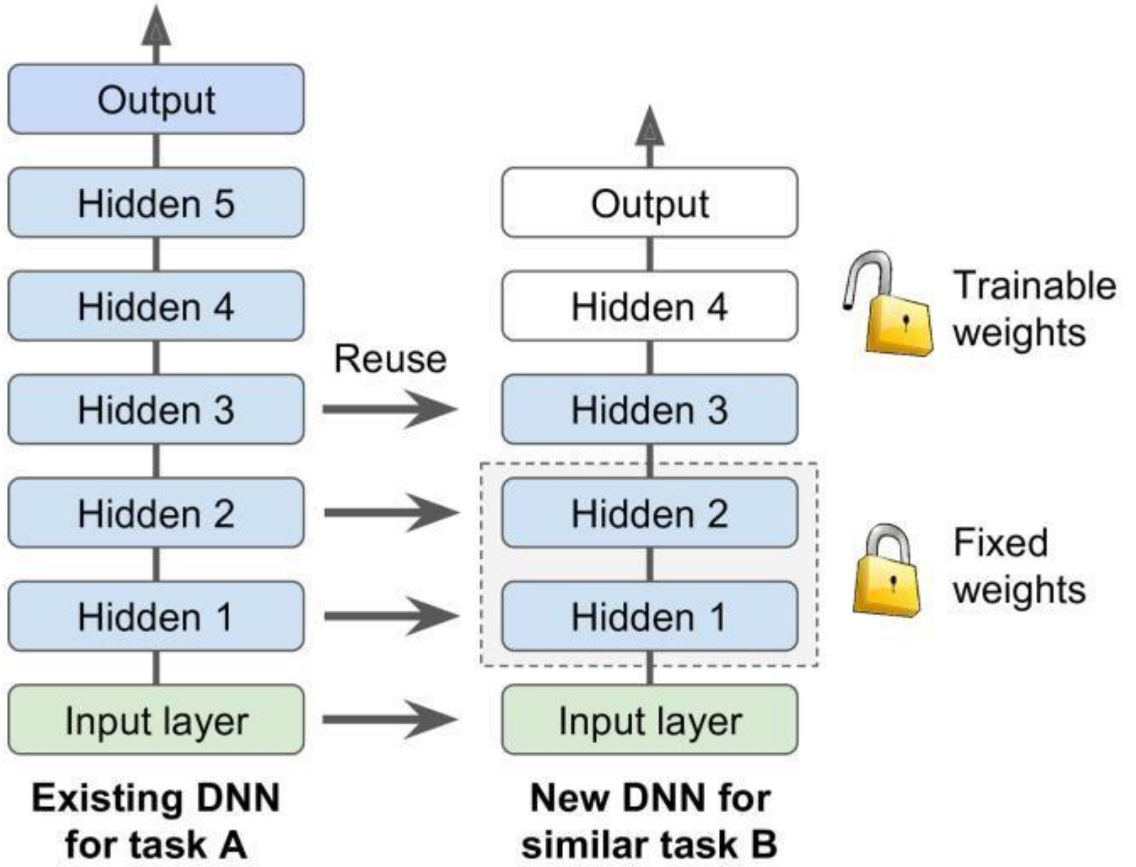
model_evaluate = model.evaluate(**X_test**, **y_test**)

- y_pred = model.predict(**X_new**)

Reusing Pretrained Layers

- It is generally not a good idea to train a very large DNN from scratch: instead, you should always try to find an existing neural network that accomplishes a similar task to the one you are trying to tackle then reuse the lower layers of this network.
- This technique is called **transfer learning**.
- It will not only **speed up training** considerably but also require significantly
  - **less training data**.
- The output layer of the original model **should usually be replaced** because it is most likely not useful at all for the new task, and it may not even have the right number of outputs for the new task.

Reusing Pretrained Layers



- ***Transfer learning will work best when the inputs have similar low-level features.***

Reusing Pretrained Layers

- Try freezing all the reused layers first (i.e., make their weights non-trainable so that Gradient Descent won't modify them), then train your model and see how it performs. Then try unfreezing one or two of the top hidden layers to let backpropagation tweak them and see if performance improves.

```
model_A = keras.models.load_model("my_model_A.h5")
model_B_on_A = keras.models.Sequential(model_A.layers[:-1])
model_B_on_A.add(keras.layers.Dense(1, activation="sigmoid"))

# If you want to avoid affecting model_A
model_A_clone = keras.models.clone_model(model_A)
model_A_clone.set_weights(model_A.get_weights())
```

Reusing Pretrained Layers

- The new output layer was initialized randomly it will make large errors.
- Freeze the reused layers during the first few epochs, giving the new layer some time to learn reasonable weights.

```
for layer in model_B_on_A.layers[:-1]:
    layer.trainable = False

model_B_on_A.compile( ... )
history = model_B_on_A.fit( ..., epochs = 5, ... )

for layer in model_B_on_A.layers[:-1]:
    layer.trainable = True

model_B_on_A.compile( ... )
history = model_B_on_A.fit( ... )
model_B_on_A.evaluate( ... )
```

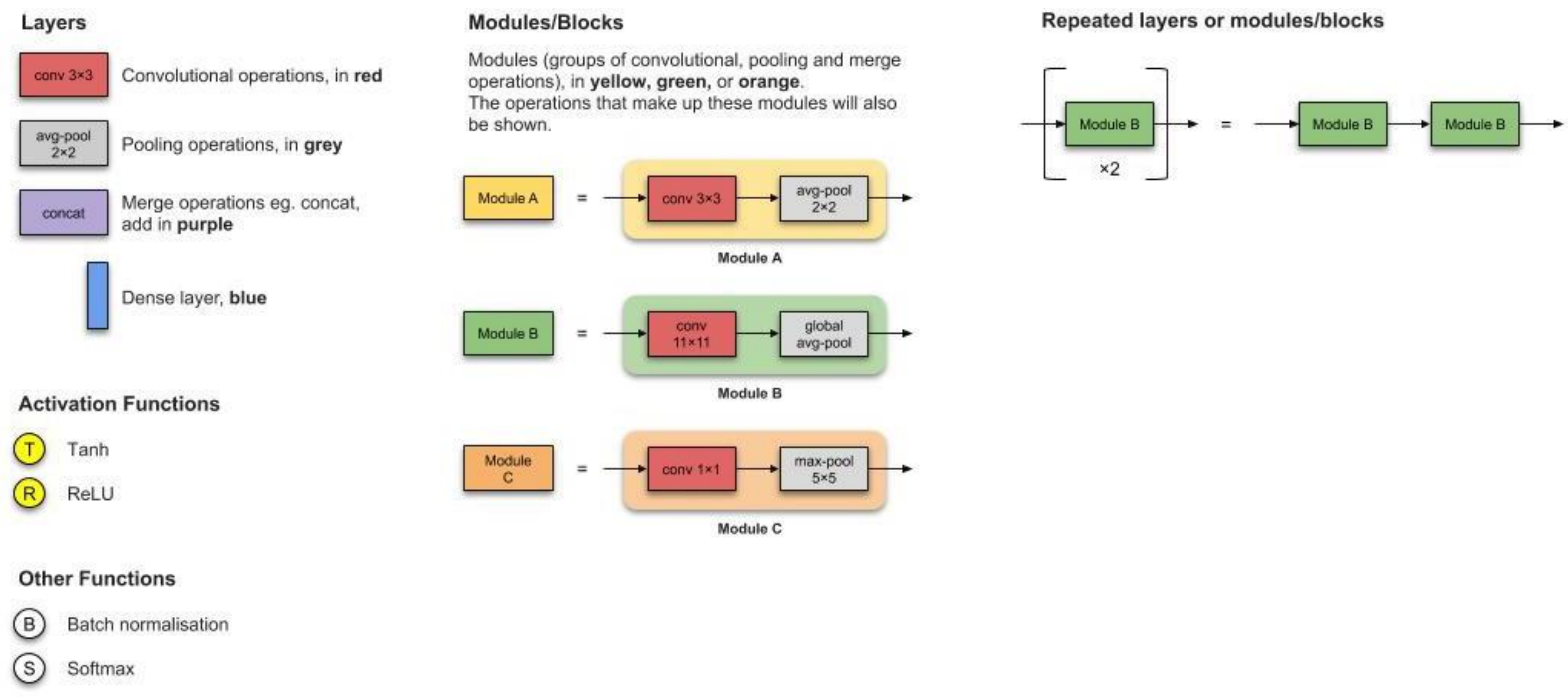
CNN Variations

Over the years, variants of the **CNN architecture** have been developed, leading to amazing advances in the field. A good measure of this progress is the error rate in competitions such as the ILSVRC **ImageNet challenge**.

1. LeNet-5 (1998)
2. AlexNet (2012)
3. VGG-16 (2014)
4. Inception-v1
5. Inception-v3
6. ResNet-50
7. Xception (2016)
8. Inception-v4 (2016)
9. Inception-ResNets
10. ResNeXt-50 (2017)

https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

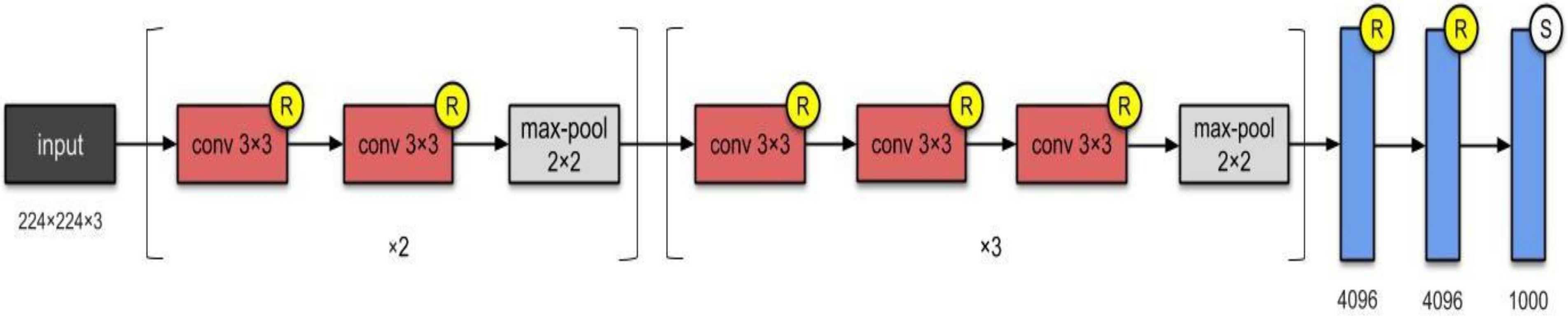
CNN Variations - Legend



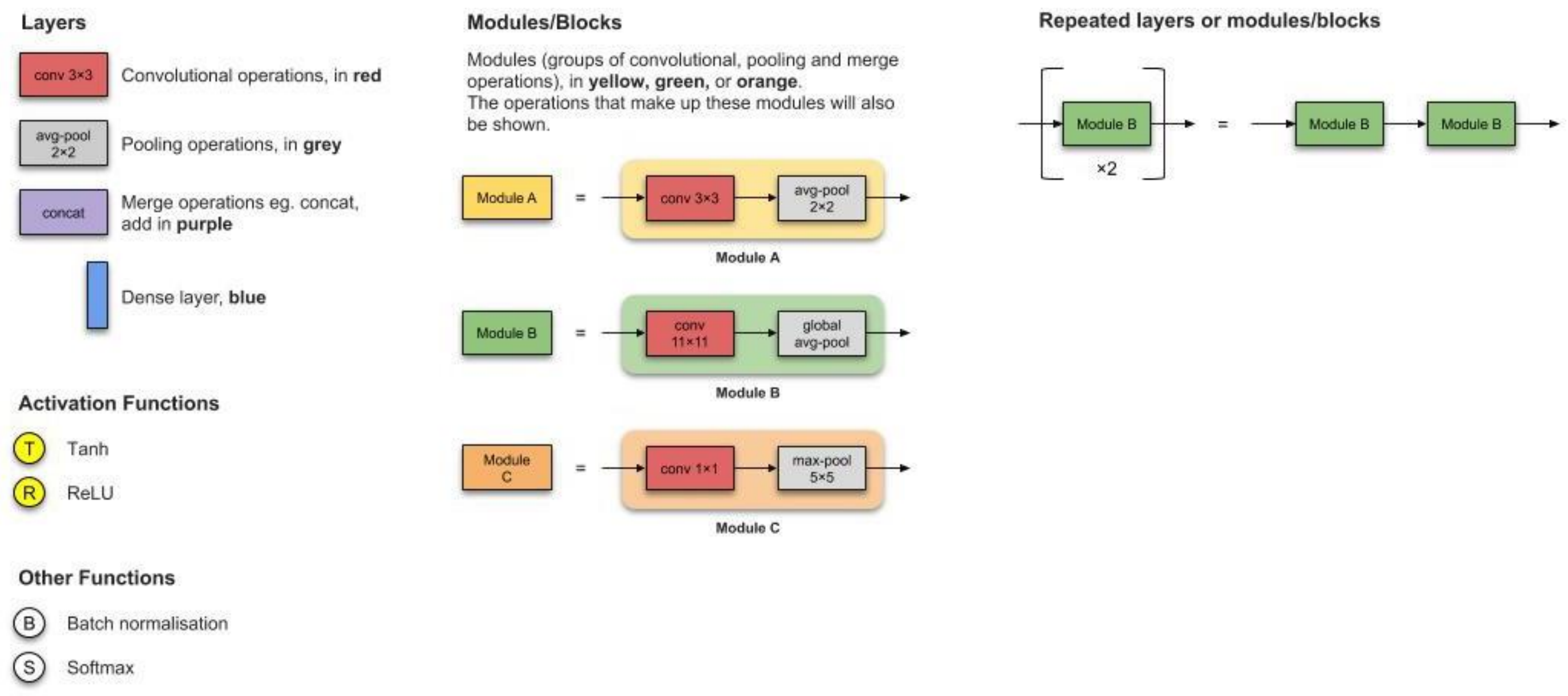
CNN Variations - VGG

- The runner-up in the ILSVRC 2014 challenge was VGG, developed by Karen Simonyan and Andrew Zisserman from the Visual Geometry Group (VGG) research lab at Oxford University.

- VGG-16 architecture and VGG-19 architecture

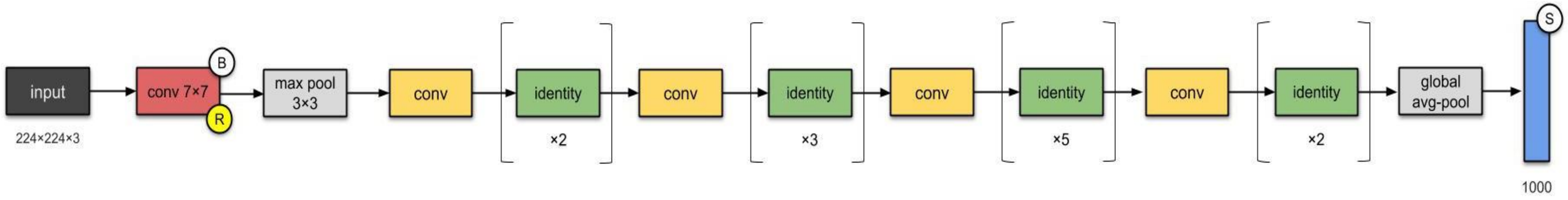


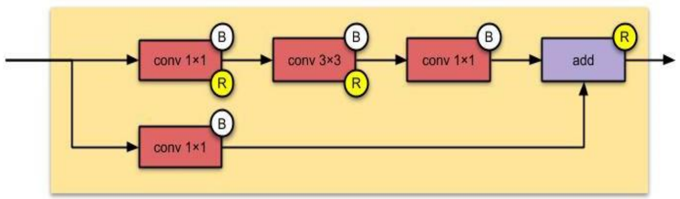
CNN Variations - Legend



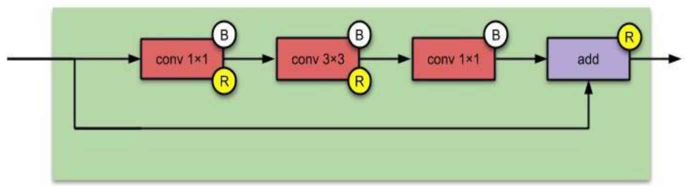
CNN Variations - ResNet-50

- The basic building block for ResNets are the conv and identity blocks.
- It uses skip connections (also called shortcut connections).
- If you add many skip connections, the network can start making progress even if several layers have not started learning yet.





Conv block



Identity block

Using Pretrained Models from Keras

Documentation for individual models

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

Using Pretrained Models from Keras

```
model = keras.applications.resnet50.ResNet50(weights="imagenet")

# you first need to ensure that the images have the right size
# ResNet-50(224 × 224)
images_resized = tf.image.resize(images, [224, 224])

# Each model provides a preprocess_input() function
inputs = keras.applications.resnet50.preprocess_input(images_resized * 255)

# Now we can use the pretrained model to make predictions
Y_proba = model.predict(inputs)
```

Using Pretrained Models from Keras

```
import tensorflow as tf
from tensorflow import keras
model = keras.applications.vgg16.VGG16(weights=None)

# The model's summary() method displays all the model's layers
print(model.summary())
```

include_top: whether to include the top layers of the network or not (**False, True**).
weights: one of **None** (random initialization) or **'imagenet'** (pre-training on ImageNet).

Pretrained Models for Transfer Learning

- If you want to build an image classifier but you do not have **enough training data**, then it is often a good idea to **reuse the lower layers of a pretrained model**.

- **For example Xception model**, we exclude the top of the network by setting **include_top=False**: this excludes the global average pooling layer and the dense output layer. We then add our own layers. Finally, we create the Keras Model:

Pretrained Models for Transfer Learning

```
base_model = keras.applications.xception.Xception(weights="imagenet",include_top=False)
avg = keras.layers.GlobalAveragePooling2D()(base_model.output)
output = keras.layers.Dense(n_classes, activation="softmax")(avg)
model = keras.Model(inputs=base_model.input, outputs=output)

for layer in base_model.layers:
    layer.trainable = False

optimizer = keras.optimizers.SGD(lr=0.2, momentum=0.9, decay=0.01)
model.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer, metrics=["accuracy"])
history = model.fit(train_set, epochs=5, validation_data=valid_set)

for layer in base_model.layers:
    layer.trainable = True

optimizer = keras.optimizers.SGD(lr=0.01, momentum=0.9, decay=0.001)
model.compile(...)
history = model.fit(...)
```

Assignment

- Use the pre-ained VGG-16 model without the dense layers.
- Add your own dense layers and classification layer.
- Choose a custom loss and optimizer
- Train your model on the “Dogs vs. Cats” dataset in the following link.
- https://www.kaggle.com/datasets/shaunthesheep/microsoft-catsvsdogs-dataset/download?datasetVersionNumber=1
  - 25000 images (12500 for each class)
  - Take 1500 from each class (1000 train & 500 test)
  - Total train (2000 images) & Total test (1000)
  - Validation is optional
- Evaluate your model on the test portion.