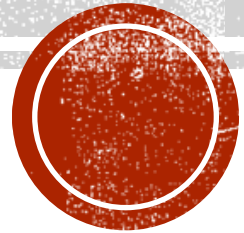
# TRANSFER LEARNING

Dr. Brian Mc Ginley



# INTRODUCTION TO TRANSFER LEARNING

- Transfer learning is a machine learning technique where a model trained on one task is reused as a starting point for a model on a different task.
- Instead of starting the learning process from scratch, transfer learning allows us to leverage knowledge gained from solving one problem to solve a different but related problem.
- Humans use transferrable skills all the time

# WHY TRANSFER LEARNING?

- **Data Efficiency:** Transfer learning enables effective learning with smaller datasets by leveraging knowledge from larger datasets.
- **Faster Training:** Starting with pre-trained weights reduces training time significantly.
- **Improved Performance:** Transfer learning often leads to better performance, especially in domains where labelled data is scarce.

# HOW TRANSFER LEARNING WORKS

- **Step 1:** Pre-train a model on a large dataset with a source task.
- **Step 2:** Fine-tune the pre-trained model on a smaller dataset with a target task.
- **Step 3:** Optionally, adapt the model architecture or add new layers for the target task.

# ADDITIONAL RESOURCES

- Keras Documentation:
  - https://keras.io/guides/transfer_learning/
- TensorFlow Tutorials:
  - https://www.tensorflow.org/tutorials/images/transfer_learning

# STEPS


Obtain the pre-trained model
Create a base model
Freeze layers
Train the new layers on the dataset
Improve the model via fine-tuning

# OBTAINING THE MODEL

- Keras pre-trained models:
- There are more than two dozen pre-trained models available from Keras.
  - https://keras.io/api/applications/

| 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 |
| ResNet152 | 232 | 76.6% | 93.1% | 60.4M | 311 | 127.4 | 6.5 |
| ResNet152V2 | 232 | 78.0% | 94.2% | 60.4M | 307 | 107.5 | 6.6 |
| InceptionV3 | 92 | 77.9% | 93.7% | 23.9M | 189 | 42.2 | 6.9 |
| InceptionResNetV2 | 215 | 80.3% | 95.3% | 55.9M | 449 | 130.2 | 10.0 |
| MobileNet | 16 | 70.4% | 89.5% | 4.3M | 55 | 22.6 | 3.4 |
| MobileNetV2 | 14 | 71.3% | 90.1% | 3.5M | 105 | 25.9 | 3.8 |
| DenseNet121 | 33 | 75.0% | 92.3% | 8.1M | 242 | 77.1 | 5.4 |
| DenseNet169 | 57 | 76.2% | 93.2% | 14.3M | 338 | 96.4 | 6.3 |

# OBTAINING THE MODEL

- Keras pre-trained models:
- There are more than two dozen pre-trained models available from Keras.
  - https://keras.io/api/applications/
- Mobile Net Example
  - That's with default values. include top means: include the fully-connected layers - often we want this as False.
  - Weights are the initial weights. If you do not use imagenet, then it will use random weights and you will need to train all these layers.

```
model = tf.keras.applications.MobileNet(
    input_shape=None,
    alpha=1.0,
    depth_multiplier=1,
    dropout=0.001,
    include_top=True,
    weights="imagenet",
    input_tensor=None,
    pooling=None,
    classes=1000,
    classifier_activation="softmax",
)
```

# OBTAINING THE MODEL

- You can also use models from TensorFlow Hub - https://www.tensorflow.org/hub

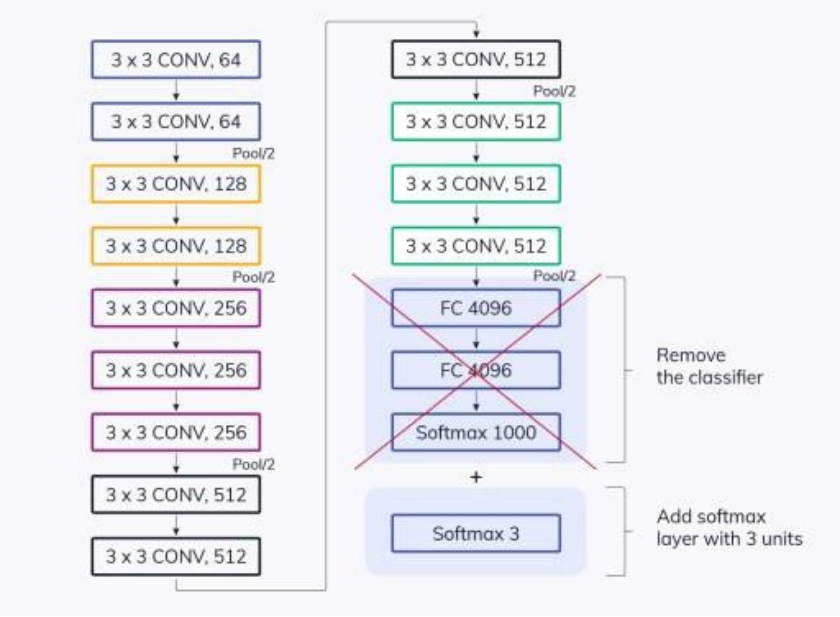
```
model = tf.keras.Sequential([
  hub.KerasLayer("https://tfhub.dev/google/imagenet/mobilenet_v2_100_224/feature_vector",
                 trainable=False),
  tf.keras.layers.Dense(num_classes, activation='softmax')
])
```

# OBTAINING THE MODEL

- You can also use models from TensorFlow Hub - https://www.tensorflow.org/hub
- There are pretrained word-embeddings available, e.g. Google's word2vec
  - https://code.google.com/archive/p/word2vec/
- And for text-processing tasks there are pretrained models at
  - https://github.com/huggingface
  - Stanza another option https://github.com/stanfordnlp/stanza/
- Lots of pre-trained models also at: https://www.kaggle.com/models
  - Inception
  - VGG
  - ResNet
  - EfficientNet
  - MobileNet
  - YOLO etc.

# CREATE BASE MODEL

- You instantiate the base model with pre-trained weights.
- The base model will usually have more units in the final output layer than you require so you have to at least remove the final output layer.
- Later on, you will add a final output layer that is compatible with your problem.



# FREEZING LAYERS

- You need to freeze the layers from the pre-trained.
- If you do not, the weights will be retrained which defeats the whole purpose of transfer learning!

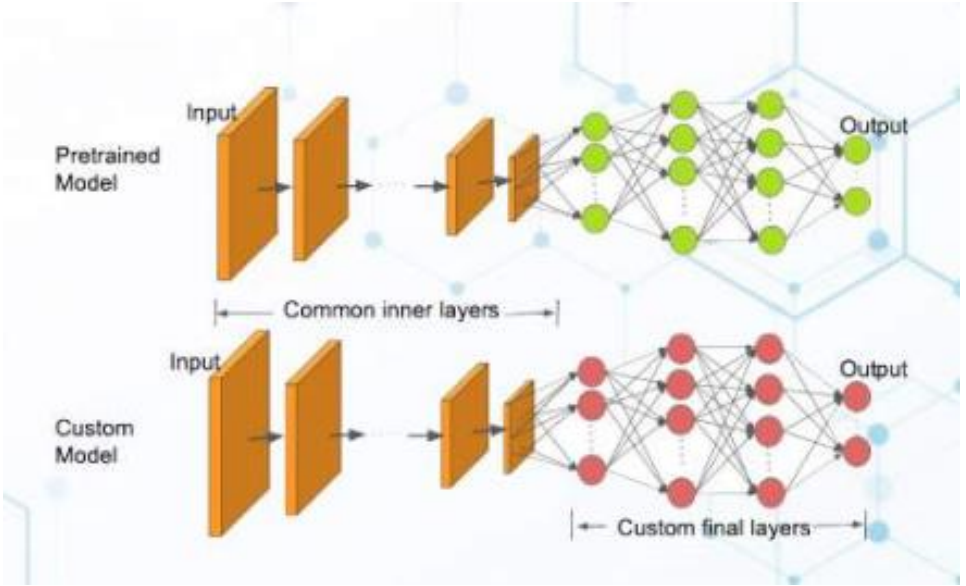
```
base_model.trainable = False
```

- You can also do it individually by layer(s)

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

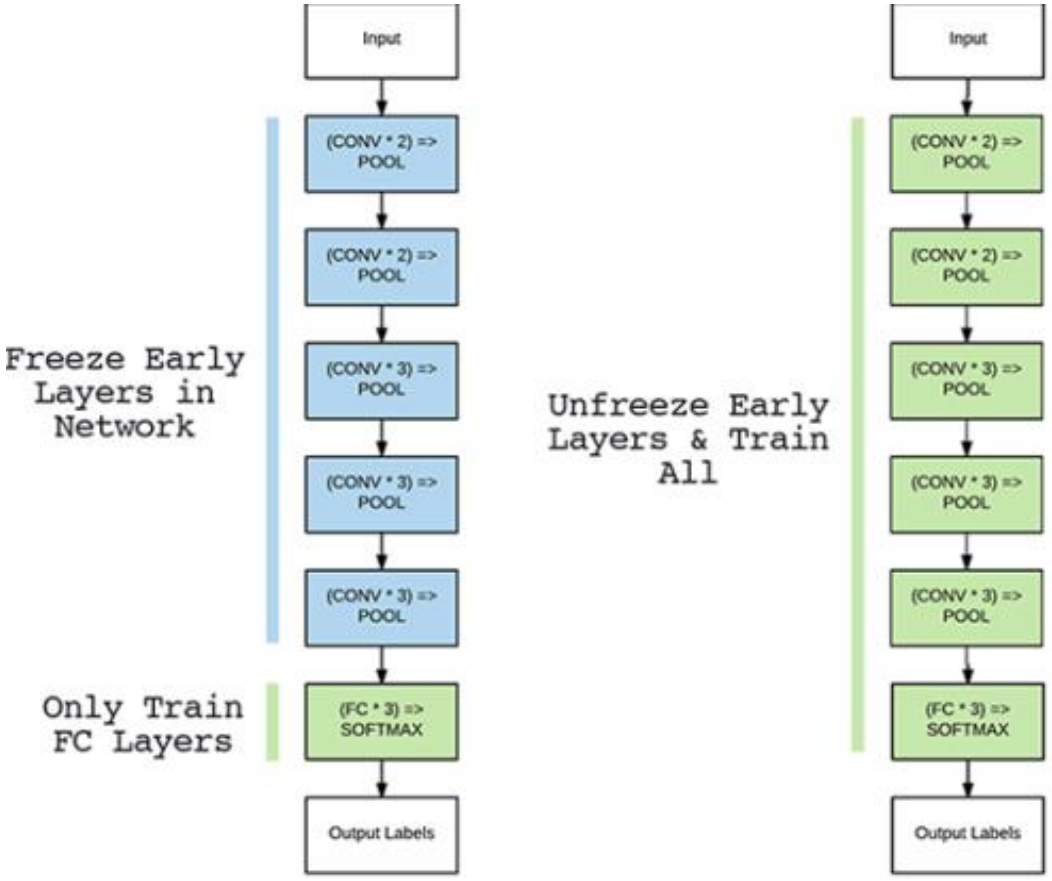
# TRAIN NEW LAYERS

- You will add some new dense layers (another hyperparameter)
- Most importantly, a final dense layer with units corresponding to the number of outputs expected by your model.
- These new layer weights are the ones you will be training.
- Make sure the output from your base model will work as the input for your new layers.
- Compile and train, you now will have a model that can make predictions on your dataset.



# FINE TUNING

- Optionally, you can improve its performance through fine-tuning.
- Fine-tuning is done by unfreezing the base model or part of it and training the entire model again on the whole dataset at a very low learning rate.



# IMPORTANT THINGS TO THINK OF

- Always pay close attention to the base model's structure and know what it is supposed to do.
- Make sure the inputs you will be sending to the base model match what is expected.
- Are the image sizes 224x224 ? Or something else? Or is the model flexible enough to do something else?
- Does the model expect data to be normalised to [0,1] or [-1,1] or something else. Often the keras application will have a preprocess input function so read the manual.

# IMPORTANT THINGS TO THINK OF

- Data Augmentation often is helpful. You may want to do this to your dataset, or you can build it into your model as before.

```
data_augmentation = tf.keras.Sequential([
  keras.layers.preprocessing.RandomFlip("horizontal"),
  keras.layers.preprocessing.RandomRotation(0.1),
])
```

# EXAMPLE

- This is taken from https://keras.io/guides/transfer_learning/, using the Xception model and some of my own alterations that I thought may help
- Instantiate the base model

```
base_model = tf.keras.applications.Xception(
weights='imagenet',
input_shape=(150, 150, 3),
include_top=False)  # Do not include the ImageNet classifier at the top
```

- Freeze the base model

```
base_model.trainable = False
```

# EXAMPLE

- Create a new model on top, first let's set the size of the input images

```
inputs = tf.keras.Input(shape=(150, 150, 3))
```

- If you want to include data augmentation as part of the model you can do it here, https://www.tensorflow.org/guide/keras/transfer_learning includes it as part of the model while the tutorial I'm following does not.

```
x = data_augmentation(inputs)
```

# EXAMPLE

- Make sure the data has been normalised as expected [-1,1] or [0,1] etc, use the function to aid you

```
# If you did not have the above data augmentation step
x = tf.keras.applications.xception.preprocess_input(inputs)
# If you did
x = tf.keras.applications.xception.preprocess_input(x)
```

- Alternatively, you can use a rescaling layer, for Xception

```
scale_layer = keras.layers.Rescaling(scale=1/127.5, offset=1)
x = scale_layer(inputs)
```

# EXAMPLE

- Ensure that the base model is running in inference mode so that batch normalization layers are not updated during the fine-tuning stage (set 'training=False')
- Convert features from the base model to vectors, using 'GlobalAveragePooling2D' (a Flatten layer may work here too)
- Add whatever Dense layers, most importantly the final dense layer - this can have a softmax activation function
- We could add Dropout here too
- Finally put the model together

# EXAMPLE

```
# We make sure that the base model is running in inference mode here,
# by passing `training=False`. This is important for fine-tuning
x = base_model(x, training=False)

# Convert features of shape `base_model.output_shape[1:]` to vectors
x = tf.keras.layers.GlobalAveragePooling2D()(x)

# A Dense classifier with 10 units
outputs = tf.keras.layers.Dense(10, activation='softmax')(x)

model = tf.keras.Model(inputs, outputs)
```

# EXAMPLE

- The previous needs to compile and fit (as done previously). Now after doing that we want to fine-tune the model we then do:

```
base_model.trainable = True
```

- Sets base model now to trainable, that's all layers. You could do individual layers by

```
freeze_layers_up_to = 10
for layer in base_model.layers[freeze_layers_up_to:]:
    layer.trainable = True
```

- You need to compile the model again, this time with a low learning rate

```
model.compile(
    optimizer=keras.optimizers.Adam(1e-5),  # Low learning rate
    loss=keras.losses.CategoricalCrossentropy(from_logits=False),
    metrics=['accuracy'],
)
```