Method 1:

- We can directly change the activation function from the main source code of the project.
- Let us check for the code where EfficientNetB2 is present.

```
def EfficientNetB2(
    include_top=True,
    weights="imagenet",
    input_tensor=None,
    input_shape=None,
    pooling=None,
    classes=1000,
    classifier_activation="softmax",
    **kwargs
):
    return EfficientNet(
        1.1,
        1.2,
        260,
        0.3,
        model_name="efficientnetb2",
        include_top=include_top,
        weights=weights,
        input_tensor=input_tensor,
        input_shape=input_shape,
        pooling=pooling,
        classes=classes,
        classifier_activation=classifier_activation,
        **kwargs
    )
```

(src: https://github.com/keras-team/keras/tree/v2.10.0/keras/applications/efficientnet.py#L646-L674)

- EfficientNetB0 to EfficientNetB7 have different number of sub-blocks and the number of sub-blocks increases as we move from B0 to B7.
- Let us check where the main architecture for the model is defined.

```
def EfficientNet(
    width_coefficient,
    depth_coefficient,
    default_size,
    dropout_rate=0.2,
    drop_connect_rate=0.2,
    depth_divisor=8,
    activation="swish",
    blocks_args="default",
    model_name="efficientnet",
    include_top=True,
    weights="imagenet",
    input_tensor=None,
    input_shape=None,
    pooling=None,
    classes=1000,
    classifier_activation="softmax",
):
    """Instantiates the EfficientNet architecture using given scaling coefficients.
```

- We can change the activation from “swish” to “relu” so that layers.activation points to tf.keras.activations.relu

```
x = layers.BatchNormalization(axis=bn_axis, name="top_bn")(x)
x = layers.Activation(activation, name="top_activation")(x)
```

- As we know from the architecture that each efficientnet model has different number of sub-blocks.
- If we dig deeper into the main “EfficientNet” method we can find a piece of code that builds those blocks and points to “block” method.

```
# Build blocks
blocks_args = copy.deepcopy(blocks_args)

b = 0
blocks = float(sum(round_repeats(args["repeats"]) for args in blocks_args))
for (i, args) in enumerate(blocks_args):
    assert args["repeats"] > 0
    # Update block input and output filters based on depth multiplier.
    args["filters_in"] = round_filters(args["filters_in"])
    args["filters_out"] = round_filters(args["filters_out"])

    for j in range(round_repeats(args.pop("repeats"))):
        # The first block needs to take care of stride and filter size
        # increase.
        if j > 0:
            args["strides"] = 1
            args["filters_in"] = args["filters_out"]
        x = block(
            x,
            activation,
            drop_connect_rate * b / blocks,
            name="block{}{}_".format(i + 1, chr(j + 97)),
            **args
        )
        b += 1
```

- In the block method we can change the activation from swish to relu so that layers.activation points to tf.keras.activations.relu

```
def block(
    inputs,
    activation="swish",
    drop_rate=0.0,
    name="",
    filters_in=32,
    filters_out=16,
    kernel_size=3,
    strides=1,
    expand_ratio=1,
    se_ratio=0.0,
    id_skip=True,
):
```

- After making changes to the model, we can retrain with our desired changes.

Method 2:

- Instead of retraining the whole model we can get the pre-trained model, change the activation layers to relu and save the model.
- Let us load the pre-trained model

```
import numpy as np
import tensorflow as tf

model = tf.keras.applications.EfficientNetB2(
    include_top=True,
    input_shape=(260, 260, 3),
    classes=1000,
    classifier_activation='softmax',
    pooling=None,
    weights='imagenet',
    input_tensor=None
 )
```

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb2.h5
37432240/37432240 [==============================] - 2s 0us/step
```

- switch_activation_layers function iterates through each layer in our model and prints out the name for each layer.

```
In [18]: def switch_activation_layers(model):
             for layer in model.layers:
                 layer_type = type(layer).__name__
                 print(layer_type)
```

```
In [19]: switch_activation_layers(model)
```

```
InputLayer
Rescaling
Normalization
Rescaling
ZeroPadding2D
Conv2D
BatchNormalization
Activation
DepthwiseConv2D
BatchNormalization
Activation
GlobalAveragePooling2D
Reshape
Conv2D
Conv2D
Multiply
Conv2D
BatchNormalization
DepthwiseConv2D
```

- Now for each layer we check for the attributes and find the activation layer which has “swish” as it’s value and replace it with relu.

```
In [41]: def switch_activation_layers(model):
             for layer in model.layers:
                 layer_type = type(layer).__name__
                 if hasattr(layer, 'activation') and layer.activation.__name__ == 'swish':
                         layer.activation = tf.keras.activations.relu
                         print(layer_type, layer.activation.__name__)
             return model
```

```
In [42]: model = switch_activation_layers(model)
```

```
Activation relu
Activation relu
Conv2D relu
Activation relu
Conv2D relu
Activation relu
Activation relu
Conv2D relu
Activation relu
Activation relu
Conv2D relu
Activation relu
Activation relu
Conv2D relu
Activation relu
Activation relu
Conv2D relu
Activation relu
Activation relu
Conv2D relu
Activation relu
Activation relu
Conv2D relu
Activation relu
Activation relu
Conv2D relu
```

- Lastly we compile the model and save it.