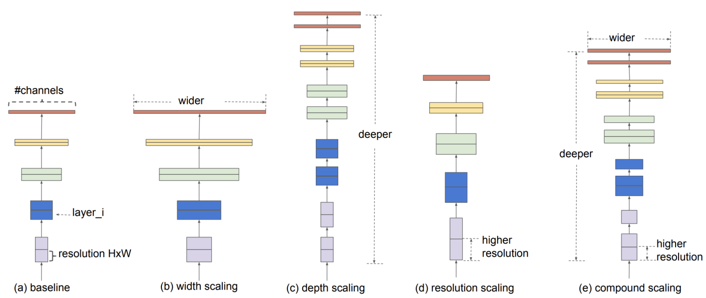
Student Name: Arash Sajjadi
Student ID: 400422096

Artificial Neural Network (ANN)
Assignment III

Question 1

EfficientNet is a convolutional neural network architecture and scaling method that uniformly scales all depth[1]/width[2]/resolution[3] dimensions using a compound coefficient. Unlike conventional practice that arbitrarily scales these factors, the EfficientNet scaling method uniformly scales network width, depth, and resolution with a set of fixed scaling coefficients. For example, we want to use 2^N times more computational resources. In that case, we can increase the network depth by α^N , width by β^N, and image size by γ^N, where α, β, and γ are constant coefficients determined by a small grid search on the original miniature model. EfficientNet uses a compound coefficient PHI to uniformly scales network width, depth, and resolution in a principled way.



The intuition justifies the compound scaling method that if the input image is more extensive, the network needs more layers to increase the receptive field and more channels to capture more fine-grained patterns on the bigger picture.

The base EfficientNet-B0 network is based on the inverted bottleneck residual blocks of MobileNetV2 and squeeze-and-excitation blocks.

EfficientNets also transfer well and achieve state-of-the-art accuracy on CIFAR-100 (91.7%), Flowers (98.8%), and three other transfer learning datasets, with an order of magnitude fewer parameters.

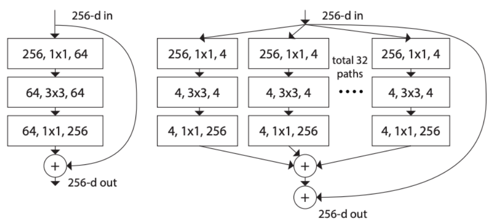
[1]Number of layers (including output but excluding input. E.g., 101). The deeper the network is, the more likely it will experience exploding or vanishing gradients, but it would be more complex and maybe more performant.

[2]Highest number of convolution kernels (channels). As pointed out by Zagoruyko and Komodakis, "wider networks tend to be able to capture more fine-grained features and are easier to train." However, a model too wide and too shallow would have difficulties in capturing higher-level features (e.g., 1024).

[3]Input image's dimension (image height * image width. e.g. 256 x 256). The higher the resolution, the more likely CNNs will be to capture fine-grained patterns, but the accuracy gain diminishes for very high resolutions (e.g., 560 x 560)).

Question 2

ResNeXt is a homogeneous neural network that reduces the number of hyperparameters required by conventional ResNet. This is achieved by using "cardinality", an additional dimension on top of the width and depth of ResNet. Cardinality defines the size of the set of transformations.

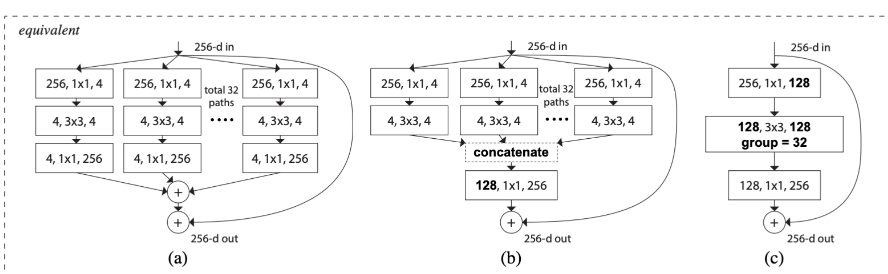


A ResNeXt repeats a building block that aggregates a set of transformations with the same topology. Compared to a ResNet, it exposes a new dimension, cardinality (the size of the set of transformations), as an essential factor in addition to depth and width dimensions.

Two rules define the basic architecture of ResNeXt. First, if the blocks produce same-dimensional spatial maps, they share the same set of hyperparameters, and if at all the spatial map is downsampled by a factor of 2, the block's width is multiplied by a factor of 2.

stage	output	ResNet-50	**ResNeXt-50 (32×4d)**
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2	3×3 max pool, stride 2
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, C{=}32 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, C{=}32 \\ 1\times1, 512 \end{bmatrix} \times 4$
conv4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, C{=}32 \\ 1\times1, 1024 \end{bmatrix} \times 6$
conv5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, C{=}32 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		$\mathbf{25.5}\times10^6$	$\mathbf{25.0}\times10^6$
FLOPs		$\mathbf{4.1}\times10^9$	$\mathbf{4.2}\times10^9$

Denote the residual block structures, whereas the numbers written adjacent to them refer to the number of stacked blocks. 32 precisely denotes that there are 32 groups in the grouped convolution.



The above network structures explain a grouped convolution and how it trumps the other two network structures.

- denotes a usual ResNeXt block that has already been seen previously. It has a cardinality of 32 and follows the split-transform-merge strategy.
- does seem to be a leaf taken out of Inception-ResNet. However, Inception or Inception-ResNet doesn't have network blocks following the same topology.
- is related to the grouped convolution, which has been proposed in AlexNet architecture. 32*4, as has been seen in (a) and (b), has been replaced with 128 in short, meaning splitting is done by a grouped convolutional layer. Similarly, the transformation is done by the other grouped convolutional layer that does 32 groups of convolutions. Later, concatenation happens.

Inception increases the network space from which the best network is to be chosen via training. Each inception module can capture salient features at different levels. Global attributes are captured by the 5x5 Conv layer, while the 3x3 Conv layer is prone to capturing distributed features. The max-pooling operation captures low-level features that stand out in a neighborhood. All of these features are extracted and concatenated at a given level before it is fed to the next layer. We leave for the network/training to decide what features hold the most value and weight accordingly. Say if the images in the data set are rich in global features without too many low-level features, then the trained Inception network will have minimal weights corresponding to the 3x3 Conv kernel compared to the 5x5 Conv kernel.

In the table below, these four CNNs are sorted w.r.t their top-5 accuracy on the Imagenet dataset. The number of trainable parameters and the Floating Point Operations (FLOP) required for a forward pass can also be seen.

As a last point, while Inception focuses on computational cost, ResNeXt focuses on computational accuracy. Intuitively, deeper networks should not perform worse than shallower networks. Still, in practice, the deeper networks performed worse than the more superficial ones, caused by overfitting and an optimization problem.

Comparison					
Network	Year	Salient Feature	top5 accuracy	Parameters	FLOP
AlexNet	2012	Deeper	84.70%	62M	1.5B
VGGNet	2014	Fixed-size kernels	92.30%	138M	19.6B
Inception	2014	Wider - Parallel kernels	93.30%	6.4M	2B
ResNet-152	2015	Shortcut connections	95.51%	60.3M	11B

Question 3

1 Natural scene classification

[1]:
```
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.colab import output
print(tf.version.VERSION)
```

```
2.9.2
```

[2]:
```
print('Default GPU Device:  {}'.format(tf.test.gpu_device_name()))
```

```
Default GPU Device:  /device:GPU:0
```

2 Setting up Random Seed

[3]:
```
RANDOM_SEED: int = 42
```

[4]:
```
import random
import os
os.environ['PYTHONHASHSEED'] = str(RANDOM_SEED)
random.seed(RANDOM_SEED)
tf.random.set_seed(RANDOM_SEED)
tf.experimental.numpy.random.seed(RANDOM_SEED)
np.random.seed(RANDOM_SEED)
```

3 Get class names (labels)

[5]:
```
!kaggle datasets download -d puneet6060/intel-image-classification
!unzip intel-image-classification.zip
output.clear()
```

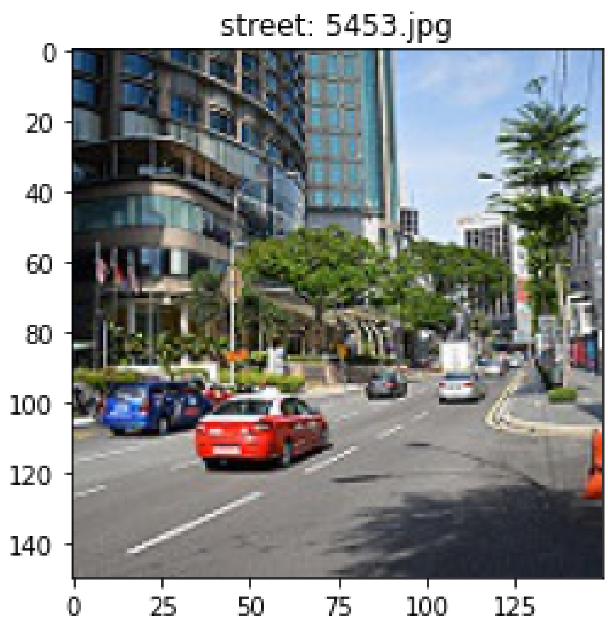
[6]:
```
import pathlib
data_dir = pathlib.Path("/content/seg_train/seg_train/")
class_names = np.array(sorted([item.name for item in data_dir.glob("*")])).
 ↪tolist() # created a list of class_names from the subdirector
class_names
```

[6]:
```
['buildings', 'forest', 'glacier', 'mountain', 'sea', 'street']
```

4 Visualize random image form dataset

[7]:
```
# visualize random image from train set
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

target_class = random.choice(class_names)
target_folder: str = "/content/seg_train/seg_train/" + target_class

# Get a random image path
random_image = random.sample(os.listdir(target_folder), 1)

# Read in the image and plot it using matplotlib
img = mpimg.imread(target_folder + "/" + random_image[0])
plt.imshow(img)
plt.title(f"{target_class}: {random_image[0]}")

print(f"Image shape: {img.shape}")
```

```
Image shape: (150, 150, 3)
```



5 Setting up Hyperparameters

[8]:
```
BATCH_SIZE: int = 32
EPOCHS: int = 12
IMAGE_SIZE = (150, 150)
AUGMENTATION_FACTOR: float = 0.2
LABEL_MODEL: str = "categorical"

TRAIN_DIR: str = "/content/seg_train/seg_train"
TEST_DIR: str = "/content/seg_test/seg_test"
```

6 Preprocessing data

[9]:
```
print("Training data  :")
train_data = tf.keras.preprocessing.image_dataset_from_directory(
    directory=TRAIN_DIR,
    label_mode=LABEL_MODEL,
    image_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    seed=RANDOM_SEED,
    shuffle=True
)

print("Testing data  :")
test_data = tf.keras.preprocessing.image_dataset_from_directory(
    directory=TEST_DIR,
    label_mode=LABEL_MODEL,
    image_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    seed=RANDOM_SEED,
    shuffle=False
)
```

```
Training data  :
Found 14034 files belonging to 6 classes.
Testing data  :
Found 3000 files belonging to 6 classes.
```

7 Visualize augmentation layer

[10]:
```
from tensorflow.keras.layers import Dense, RandomFlip, RandomRotation,␣
 ↪RandomZoom, RandomWidth, RandomHeight, Rescaling
from tensorflow.keras import Sequential
from tensorflow.keras.activations import softmax
```

[11]:
```
augmentation_layer = Sequential([
    RandomFlip("horizontal"),
    RandomRotation(AUGMENTATION_FACTOR),
    RandomZoom(AUGMENTATION_FACTOR),
    RandomWidth(AUGMENTATION_FACTOR),
    RandomHeight(AUGMENTATION_FACTOR),
    Rescaling(1./255)
], name="augmentation_layer")

augmentation_layer
```

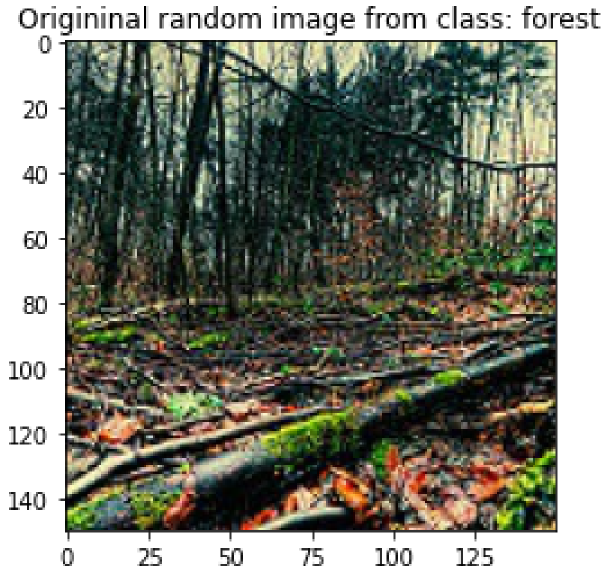
[11]:
```
<keras.engine.sequential.Sequential at 0x7f61dc305e80>
```

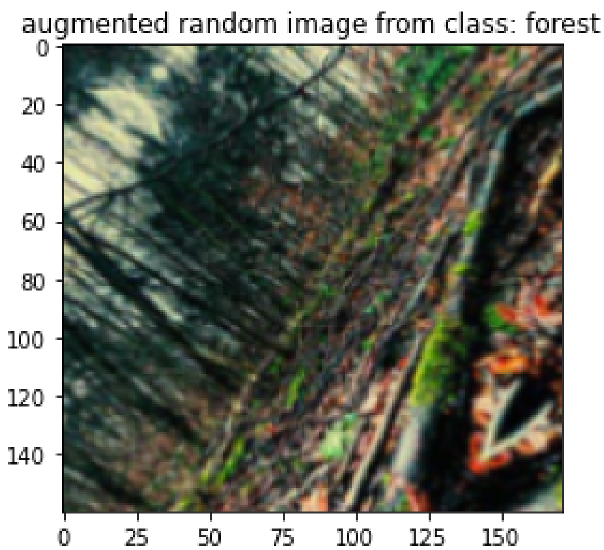
8 Visualize augmentation layer

[12]:
```
target_dir = TRAIN_DIR
target_class = random.choice(class_names)
target_dir = f"{target_dir}/{target_class}"
random_image = random.choice(os.listdir(target_dir))
random_image_path = target_dir + "/" + random_image

print(random_image_path)
# Read in the random image
img = mpimg.imread(random_image_path)
plt.title(f"Origininal random image from class: {target_class}")
# plt.axis(False)
plt.imshow(img);

# Now lets plot our augmented random image
augmented_image = augmentation_layer(img, training=True)
plt.figure()
plt.title(f"augmented random image from class: {target_class}")
plt.imshow(augmented_image)
```

```
/content/seg_train/seg_train/forest/2923.jpg
```

[12]:
```
<matplotlib.image.AxesImage at 0x7f61dc244730>
```


Origininal random image from class: forest
0
20
40
60
80
100
120
140
0
25
50
75
100
125


augmented random image from class: forest
0
20
40
60
80
100
120
140
0
25
50
75
100
125
150

9 Creating teh backbone base model using ResNet50V2

[13]:
```
# Turn off all warnings except for errors
tf.get_logger().setLevel('ERROR')

# set seed
tf.random.set_seed(RANDOM_SEED)

base_model = tf.keras.applications.ResNet50V2(include_top=False)
base_model.trainable = False

# Creating input layer
input_layer = tf.keras.layers.Input(shape=IMAGE_SIZE + (3,), name="input_layer")
x = augmentation_layer(input_layer)
x = base_model(x, training=False)
x = tf.keras.layers.
 ↪GlobalAveragePooling2D(name="global_average_pooling_2d_layer")(x)
output_layer = Dense(len(class_names), activation=softmax,␣
 ↪name="output_layer")(x)

model_1 = tf.keras.Model(input_layer, output_layer)
```

[14]:
```
model_1.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_layer (InputLayer)    [(None, 150, 150, 3)]     0

 augmentation_layer (Sequent  (None, None, 3)          0
 ial)

 resnet50v2 (Functional)     (None, None, None, 2048)  23564800

 global_average_pooling_2d_l  (None, 2048)             0
 ayer (GlobalAveragePooling2
 D)

 output_layer (Dense)        (None, 6)                 12294

=================================================================
Total params: 23,577,094
Trainable params: 12,294
Non-trainable params: 23,564,800
_________________________________________________________________
```

10 Tensorflow Callbacks

[15]:
```
early_stopping_callback = tf.keras.callbacks.EarlyStopping(monitor="val_loss",
                                                          patience=3)
reduce_lr_callback = tf.keras.callbacks.ReduceLROnPlateau(monitor="val_loss",
                                                         factor=0.2,
                                                         patience=2,
                                                         verbose=1,
                                                         min_lr=1e-7)

# set checkpoint path
checkpoint_path = "checkpoint_weights/checkpoint.cpk"

# Create a ModelCheckpoint callback that saves the model's weights only
checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_path,
    save_weights_only=True,
    save_best_only=True,
    save_freq="epoch", #save every epoch
    verbose=1
)
```

11 Compile the model

[16]:
```
model_1.compile(
    loss=tf.keras.losses.CategoricalCrossentropy(),
    optimizer=tf.keras.optimizers.Adam(),
    metrics=["accuracy"]
)
```

12 Fit the model

[17]:
```
history_1 = model_1.fit(
    train_data,
    epochs=EPOCHS,
    steps_per_epoch=len(train_data),
    validation_data=test_data,
    validation_steps=int(0.15 * len(test_data)),
    callbacks=[
        early_stopping_callback,
        reduce_lr_callback,
        checkpoint_callback
    ]
)
```

```
Epoch 1/12
```

```
439/439 [==============================] - ETA: 0s - loss: 0.5573 - accuracy:
0.7953
Epoch 1: val_loss improved from inf to 0.36231, saving model to
checkpoint_weights/checkpoint.cpk
439/439 [==============================] - 100s 214ms/step - loss: 0.5573 -
accuracy: 0.7953 - val_loss: 0.3623 - val_accuracy: 0.8661 - lr: 0.0010
Epoch 2/12
439/439 [==============================] - ETA: 0s - loss: 0.4293 - accuracy:
0.8448
Epoch 2: val_loss improved from 0.36231 to 0.18632, saving model to
checkpoint_weights/checkpoint.cpk
439/439 [==============================] - 54s 124ms/step - loss: 0.4293 -
accuracy: 0.8448 - val_loss: 0.1863 - val_accuracy: 0.9241 - lr: 0.0010
Epoch 3/12
439/439 [==============================] - ETA: 0s - loss: 0.4066 - accuracy:
0.8539
Epoch 3: val_loss did not improve from 0.18632
439/439 [==============================] - 51s 115ms/step - loss: 0.4066 -
accuracy: 0.8539 - val_loss: 0.3115 - val_accuracy: 0.8906 - lr: 0.0010
Epoch 4/12
439/439 [==============================] - ETA: 0s - loss: 0.3944 - accuracy:
0.8555
Epoch 4: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.

Epoch 4: val_loss did not improve from 0.18632
439/439 [==============================] - 48s 109ms/step - loss: 0.3944 -
accuracy: 0.8555 - val_loss: 0.2292 - val_accuracy: 0.9085 - lr: 0.0010
Epoch 5/12
439/439 [==============================] - ETA: 0s - loss: 0.3550 - accuracy:
0.8724
Epoch 5: val_loss did not improve from 0.18632
439/439 [==============================] - 46s 104ms/step - loss: 0.3550 -
accuracy: 0.8724 - val_loss: 0.2607 - val_accuracy: 0.8996 - lr: 2.0000e-04
```

[18]:
```
# load in saved model wights and evaluate model
model_1.load_weights(checkpoint_path)
```

[18]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at
0x7f61c2432fa0>
```

[19]:
```
model_1.evaluate(test_data)
```

```
94/94 [==============================] - 6s 62ms/step - loss: 0.3210 - accuracy:
0.8810
```

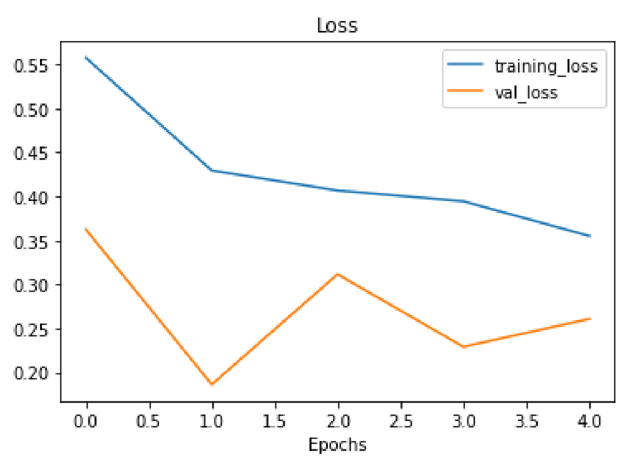
[19]:
```
[0.3210234045982361, 0.8809999823570251]
```

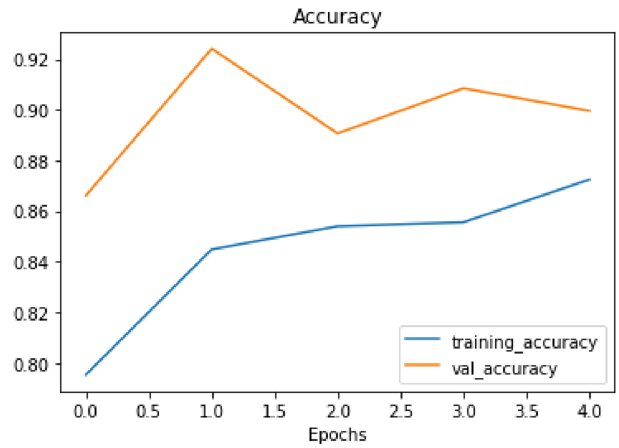
[20]:

```
import matplotlib.pyplot as plt
def plot_loss_curve(history):
    loss = history_1.history['loss']
    val_loss = history_1.history['val_loss']

    accuracy = history_1.history["accuracy"]
    val_accuracy = history_1.history["val_accuracy"]

    epochs = range(len(history_1.history['loss']))

    # Plot loss
    plt.plot(epochs, loss, label='training_loss')
    plt.plot(epochs, val_loss, label='val_loss')
    plt.title('Loss')
    plt.xlabel('Epochs')
    plt.legend()

    # Plot accuracy
    plt.figure()
    plt.plot(epochs, accuracy, label='training_accuracy')
    plt.plot(epochs, val_accuracy, label='val_accuracy')
    plt.title('Accuracy')
    plt.xlabel('Epochs')
    plt.legend()
```

[21]:

```
plot_loss_curve(history_1)
```





[22]:

```python
# Turn off all warnings except for errors
tf.get_logger().setLevel('ERROR')

# set seed
tf.random.set_seed(RANDOM_SEED)


base_model.trainable = True

for layer in base_model.layers[:-20]:
    layer.trainable = False
```

[23]:

```python
model_1.compile(
    loss=tf.keras.losses.CategoricalCrossentropy(),
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
    metrics=["accuracy"]
)
```

[24]:

```python
history_1.epoch[-1]
```

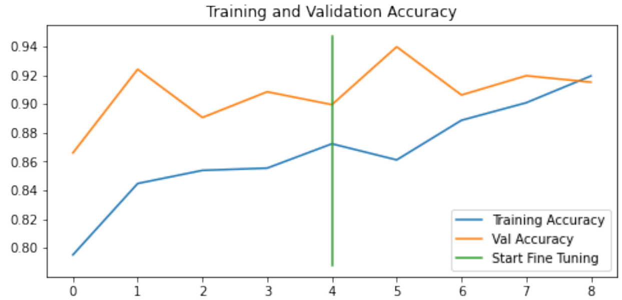
[24]: 4

[25]:
```python
# fine tune for another 5 epochs

fine_tune_epochs = EPOCHS + 10

# Refit the model (same as model_2 except with more trainable layers)
history_2 = model_1.fit(
    train_data,
    epochs=fine_tune_epochs,
    validation_data=test_data,
    steps_per_epoch=len(train_data),
    validation_steps=int(0.15 * len(test_data)),
    initial_epoch=history_1.epoch[-1],
    callbacks=[
        early_stopping_callback,
        reduce_lr_callback,
        checkpoint_callback
    ]
)
```

```
Epoch 5/22
439/439 [==============================] - ETA: 0s - loss: 0.3821 - accuracy:
0.8612
Epoch 5: val_loss improved from 0.18632 to 0.16137, saving model to
checkpoint_weights/checkpoint.cpk
439/439 [==============================] - 58s 122ms/step - loss: 0.3821 -
accuracy: 0.8612 - val_loss: 0.1614 - val_accuracy: 0.9397 - lr: 1.0000e-04
Epoch 6/22
439/439 [==============================] - ETA: 0s - loss: 0.3087 - accuracy:
0.8887
Epoch 6: val_loss did not improve from 0.16137
439/439 [==============================] - 49s 112ms/step - loss: 0.3087 -
accuracy: 0.8887 - val_loss: 0.2182 - val_accuracy: 0.9062 - lr: 1.0000e-04
Epoch 7/22
439/439 [==============================] - ETA: 0s - loss: 0.2792 - accuracy:
0.9009
Epoch 7: ReduceLROnPlateau reducing learning rate to 1.9999999494757503e-05.

Epoch 7: val_loss did not improve from 0.16137
439/439 [==============================] - 48s 108ms/step - loss: 0.2792 -
accuracy: 0.9009 - val_loss: 0.2073 - val_accuracy: 0.9196 - lr: 1.0000e-04
Epoch 8/22
439/439 [==============================] - ETA: 0s - loss: 0.2224 - accuracy:
0.9195
Epoch 8: val_loss did not improve from 0.16137
439/439 [==============================] - 48s 109ms/step - loss: 0.2224 -
accuracy: 0.9195 - val_loss: 0.2202 - val_accuracy: 0.9152 - lr: 2.0000e-05
```

[26]:
```
# Let's create a function to compare training histories
def compare_historys(original_history, new_history, initial_epochs=5, metric:␣
↪str = "accuracy"):
    """
    Compares two TensorFlow History objects.
    """
    # Get original history measurements
    acc = original_history.history[metric]
    loss = original_history.history["loss"]

    val_acc = original_history.history[f"val_{metric}"]
    val_loss = original_history.history["val_loss"]

    # Combine original history metrics with new_history metrics
    total_acc = acc + new_history.history[metric]
    total_loss = loss + new_history.history["loss"]

    total_val_acc = val_acc + new_history.history[f"val_{metric}"]
    total_val_loss = val_loss + new_history.history["val_loss"]

    # Make plot for accuracy
    plt.figure(figsize=(8, 8))
    plt.subplot(2, 1, 1)
    plt.plot(total_acc, label="Training Accuracy")
    plt.plot(total_val_acc, label="Val Accuracy")
    plt.plot([initial_epochs-1, initial_epochs-1], plt.ylim(), label="Start Fine␣
↪Tuning")
    plt.legend(loc="lower right")
    plt.title("Training and Validation Accuracy")

    # Make plot for loss
    plt.figure(figsize=(8, 8))
    plt.subplot(2, 1, 2)
    plt.plot(total_loss, label="Training Loss")
    plt.plot(total_val_loss, label="Val Loss")
    plt.plot([initial_epochs-1, initial_epochs-1], plt.ylim(), label="Start Fine␣
↪Tuning")
    plt.legend(loc="upper right")
    plt.title("Training and Validation Loss")
```

[27]:
```
compare_historys(history_1, history_2)
```



[28]:
```
# load in saved model weights and evaluate model
model_1.load_weights(checkpoint_path)
```

[28]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at
0x7f61dc04cfa0>
```

[29]:
```
model_1.evaluate(test_data)
```

```
94/94 [==============================] - 5s 54ms/step - loss: 0.2610 - accuracy:
0.9000
```

[29]:
```
[0.261038601398468, 0.8999999761581421]
```

13 Making predictions

[30]:
```
# Make predictions with model
preds_probs = model_1.predict(test_data)
preds_probs[:10]
```

```
94/94 [==============================] - 5s 50ms/step
```

```
[30]: array([[9.9990058e-01, 4.2831164e-08, 2.7188033e-05, 9.7682114e-06,
         1.0723115e-05, 5.1790419e-05],
        [9.8721343e-01, 1.4112474e-05, 9.3190138e-05, 8.5504325e-05,
         2.6390061e-04, 1.2329934e-02],
        [9.9800485e-01, 3.1354713e-07, 9.3854014e-06, 9.0478679e-05,
         7.1727110e-05, 1.8232594e-03],
        [9.9022043e-01, 1.3345740e-06, 7.9958641e-05, 1.2207507e-03,
         4.3091490e-03, 4.1683889e-03],
        [9.8196453e-01, 4.2899876e-05, 2.2451412e-03, 2.0842291e-03,
         1.2903975e-02, 7.5932709e-04],
        [3.9960101e-01, 4.8900376e-05, 2.2721787e-03, 1.4422562e-02,
         5.2975065e-01, 5.3904641e-02],
        [9.9230701e-01, 1.5337193e-05, 2.9103007e-04, 7.2559790e-04,
         7.1279053e-04, 5.9482004e-03],
        [2.4947867e-01, 8.3858497e-05, 6.9935521e-04, 8.5182366e-04,
         2.8611184e-03, 7.4602515e-01],
        [9.9140406e-01, 4.2123756e-06, 3.9994295e-04, 2.8560358e-05,
         6.1300234e-04, 7.5500770e-03],
        [9.9613476e-01, 5.0047133e-06, 5.2291594e-05, 1.3840073e-05,
         3.2360532e-04, 3.4704928e-03]], dtype=float32)
```

```
[31]: # Get the pred classes of each label
pred_classes = preds_probs.argmax(axis=1)
pred_classes[:10]
```

```
[31]: array([0, 0, 0, 0, 0, 4, 0, 5, 0, 0])
```

```
[32]: # To get our test labels we need to unravel our test_data BatchDataset
y_labels = []
for images, labels in test_data.unbatch():
  y_labels.append(labels.numpy().argmax()) # currently test labels look like:␣
↪[0, 0, 0, 1, .... 0, 0], we want the index value where the "1" occurs
y_labels[:10] # look at the first 10
```

```
[32]: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

14 Reports

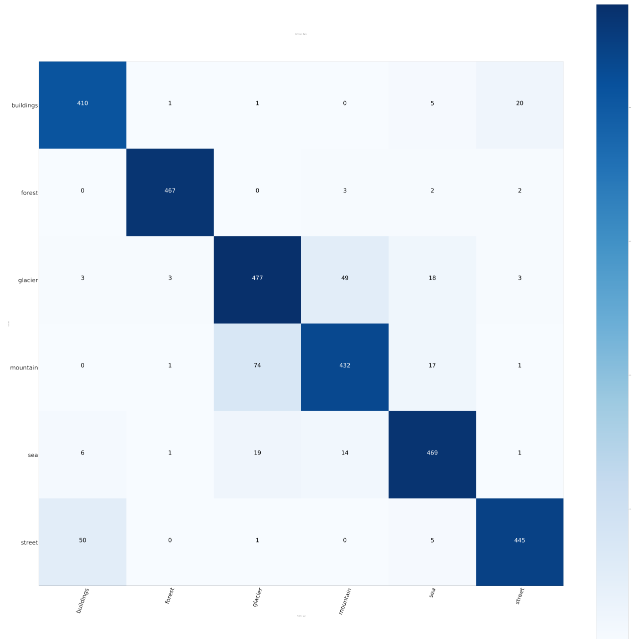
```
[33]: from sklearn.metrics import accuracy_score
sklearn_accuracy = accuracy_score(y_true=y_labels,
                                  y_pred=pred_classes)
sklearn_accuracy
```

```
[33]: 0.9
```

[34]:

```python
import itertools
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix

# We need to make some changes to our make_confusion_matrix function to ensure the x-labels print vertically
def make_confusion_matrix(y_true, y_pred, classes=None, figsize=(10, 10), text_size=15, norm=False, savefig=False):
  """Makes a labelled confusion matrix comparing predictions and ground truth labels.

  If classes is passed, confusion matrix will be labelled, if not, integer class values
  will be used.

  Args:
    y_true: Array of truth labels (must be same shape as y_pred).
    y_pred: Array of predicted labels (must be same shape as y_true).
    classes: Array of class labels (e.g. string form). If `None`, integer labels are used.
    figsize: Size of output figure (default=(10, 10)).
    text_size: Size of output figure text (default=15).
    norm: normalize values or not (default=False).
    savefig: save confusion matrix to file (default=False).

  Returns:
    A labelled confusion matrix plot comparing y_true and y_pred.

  Example usage:
    make_confusion_matrix(y_true=test_labels, # ground truth test labels
                          y_pred=y_preds, # predicted labels
                          classes=class_names, # array of class label names
                          figsize=(15, 15),
                          text_size=10)
  """
  # Create the confustion matrix
  cm = confusion_matrix(y_true, y_pred)
  cm_norm = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis] # normalize it
  n_classes = cm.shape[0] # find the number of classes we're dealing with

  # Plot the figure and make it pretty
  fig, ax = plt.subplots(figsize=figsize)
  cax = ax.matshow(cm, cmap=plt.cm.Blues) # colors will represent how 'correct' a class is, darker == better
  fig.colorbar(cax)

  # Are there a list of classes?
  if classes:
    labels = classes
  else:
    labels = np.arange(cm.shape[0])
```

[35]:
```
make_confusion_matrix(y_true=y_labels,
                      y_pred=pred_classes,
                      classes=class_names,
                      figsize=(100, 100),
                      text_size=50)
```



[36]:
```
from sklearn.metrics import classification_report
print(classification_report(y_true=y_labels,
                            y_pred=pred_classes))
```

```
              precision    recall  f1-score   support

           0       0.87      0.94      0.91       437
           1       0.99      0.99      0.99       474
           2       0.83      0.86      0.85       553
           3       0.87      0.82      0.84       525
           4       0.91      0.92      0.91       510
           5       0.94      0.89      0.91       501

    accuracy                           0.90      3000
   macro avg       0.90      0.90      0.90      3000
weighted avg       0.90      0.90      0.90      3000
```

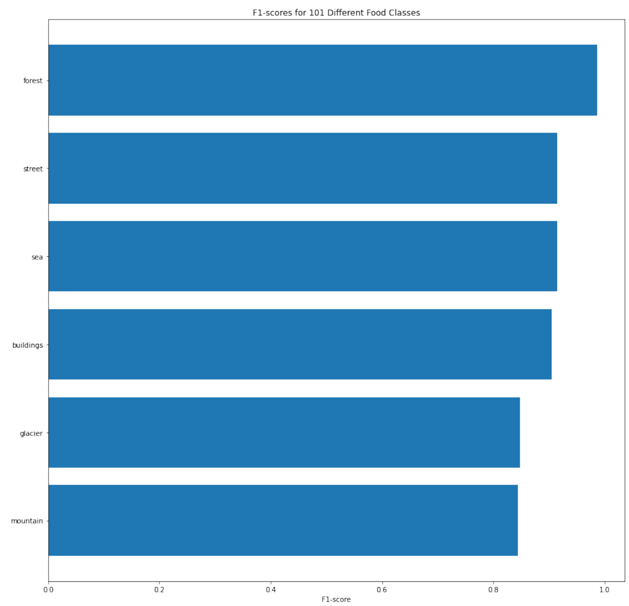
[37]:
```python
from sklearn.metrics import classification_report
import pandas as pd

def get_f1_score_on_every_class_name(y_labels, y_true, class_names):
    """Return f1 score on every class name as a dataframe

    Args:
        y_labels (_type_): y_true of test_
        y_pred (_type_): predictions list

    Returns:
        pd.DataFrame: f1-scores dataframe on every class name
    """
    classification_report_dict = classification_report(y_labels, y_true,␣
 ↪output_dict=True)
    # Create empty dictionary
    class_f1_scores = {}
    # Loop through classification report dictionary items
    for k, v in classification_report_dict.items():
        if k == "accuracy": # stop once we get to accuracy key
            break
        else:
            # Add class names and f1-scores to new dictionary
            class_f1_scores[class_names[int(k)]] = v["f1-score"]
    class_f1_scores

    # Trun f1-scores into dataframe for visualization
    f1_scores = pd.DataFrame({"class_names": list(class_f1_scores.keys()),
                              "f1-score": list(class_f1_scores.values())}).
 ↪sort_values("f1-score", ascending=False)
    return f1_scores

f1_scores = get_f1_score_on_every_class_name(y_labels=y_labels,␣
 ↪y_true=pred_classes, class_names=class_names)
f1_scores
```

[37]:

	class_names	f1-score
1	forest	0.986272
5	street	0.914697
4	sea	0.914230
0	buildings	0.905077
2	glacier	0.848000
3	mountain	0.844575

[38]:
```
import matplotlib.pyplot as plt
import pandas as pd

def plot_f1_scores_on_every_class_name(f1_scores, figsize = (10, 10)):
    fig, ax = plt.subplots(figsize=figsize)
    scores = ax.barh(range(len(f1_scores)), f1_scores["f1-score"].values)
    #ax.bar_label(scores, label_type='center', c="white")
    ax.set_yticks(range(len(f1_scores)))
    ax.set_yticklabels(f1_scores["class_names"])
    ax.set_xlabel("F1-score")
    ax.set_title("F1-scores for 101 Different Food Classes")
    ax.invert_yaxis(); # reverse the order of our plot
```

[39]:
```
plot_f1_scores_on_every_class_name(f1_scores=f1_scores, figsize=(15, 15))
```



[40]:

```python
# Create a function to load and prepare images
def load_and_prep_image(filename, img_shape=150, scale=True):
  """
  Reads in an image from filename, turns it into a tensor and reshapes into
  specified shape (img_shape, img_shape, color_channels=3).

  Args:
    filename (str): path to target image
    image_shape (int): height/width dimension of target image size
    scale (bool): scale pixel values from 0-255 to 0-1 or not

  Returns:
    Image tensor of shape (img_shape, img_shape, 3)
  """
  # Read in the image
  img = tf.io.read_file(filename)

  # Decode image into tensor
  img = tf.io.decode_image(img, channels=3)

  # Resize the image
  # img = tf.image.resize(img, [img_shape, img_shape])
  img = tf.image.resize(img, list(IMAGE_SIZE))

  # Scale? Yes/no
  if scale:
    # rescale the image (get all values between 0 and 1)
    return img/255.
  else:
    return img # don't need to rescale images for EfficientNet models in␣
 ↪TensorFlow
```

[41]:

```
# Make preds on a series of random images
import os
import random

PRED_DIR: str ="/content/seg_pred/seg_pred"
plt.figure(figsize=(17, 10))
for i in range(3):
  # Choose random image(s) from random class(es)
  filename = random.choice(os.listdir(PRED_DIR))
  filepath = PRED_DIR + "/" + filename
  print(filepath)
  # Load the image and make predictions
  img = load_and_prep_image(filepath, scale=False)
  img_expanded = tf.expand_dims(img, axis=0)
  pred_prob = model_1.predict(img_expanded)
  pred_class = class_names[pred_prob.argmax()]

  plt.subplot(1, 3, i+1)
  plt.imshow(img/225.)
  plt.title(f"pred: {pred_class}, prob: {pred_prob.max():.2f}")
  plt.axis(False);
```

```
/content/seg_pred/seg_pred/5802.jpg
1/1 [==============================] - 1s 1s/step

WARNING:matplotlib.image:Clipping input data to the valid range for imshow with
RGB data ([0..1] for floats or [0..255] for integers).

/content/seg_pred/seg_pred/20760.jpg
1/1 [==============================] - 0s 20ms/step

WARNING:matplotlib.image:Clipping input data to the valid range for imshow with
RGB data ([0..1] for floats or [0..255] for integers).

/content/seg_pred/seg_pred/19330.jpg
1/1 [==============================] - 0s 19ms/step

WARNING:matplotlib.image:Clipping input data to the valid range for imshow with
RGB data ([0..1] for floats or [0..255] for integers).
```