



# Demo python programming

---

Dr. Tushar Sandhan

## Setup : (environment)

Import TensorFlow and other necessary libraries:

```
[ ] import matplotlib.pyplot as plt  
import numpy as np  
import PIL  
import tensorflow as tf
```

```
from tensorflow import keras  
from tensorflow.keras import layers  
from tensorflow.keras.models import Sequential
```

np.sum()  
np.sum(...)

python image library

Google co-lab.

python

J

1) tensorflow

2) pytorch

keras

python notebook

programming

interface

Matlab

wrapper covering  
(functionalities)  
over tensorflow

Program/ implement CNN  
for flower type classification

e.g.  
roses  
tulips

1.jpg  
2.jpg  
3.jpg  
101.jpg  
102.jpg

dataset directory  
where images  
are stored.

```
[ ] import pathlib  
dataset_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz"  
data_dir = tf.keras.utils.get_file('flower_photos', origin=dataset_url, untar=True)  
data_dir = pathlib.Path(data_dir)
```

Downloading data from https://storage.googleapis.com/download.tensorflow.org/example\_images/flower\_photos.tgz  
228813984/228813984 [=====] - 3s 0us/step

After downloading, you should now have a copy of the dataset available. There are 3,670 total images:  
array, list, tuple

```
[ ] image_count = len(list(data_dir.glob('*/*-jpg')))
```

3670  
Training  
(validation/test)

Test → (in real world) ← Unseen Samples

Train  
(val)  
test.

Here are some roses:

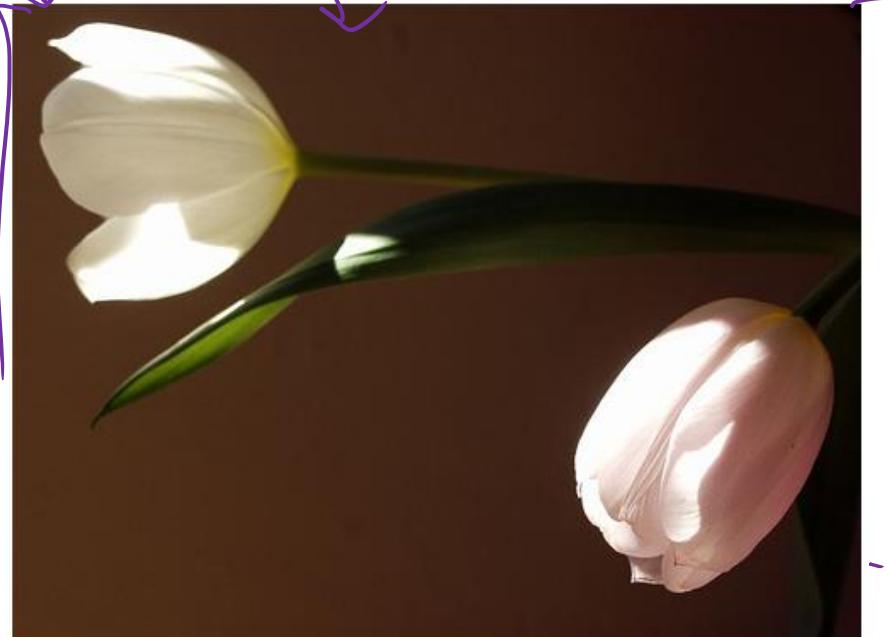
```
[ ] roses = list(data_dir.glob('roses/*'))  
PIL.Image.open(str(roses[0]))  
1.jpg
```



list of all images in roses' folder  
roses[0] → 1.jpg  
roses[1] → 2.jpg

And some tulips:

```
[ ] tulips = list(data_dir.glob('tulips/*'))  
PIL.Image.open(str(tulips[0]))
```

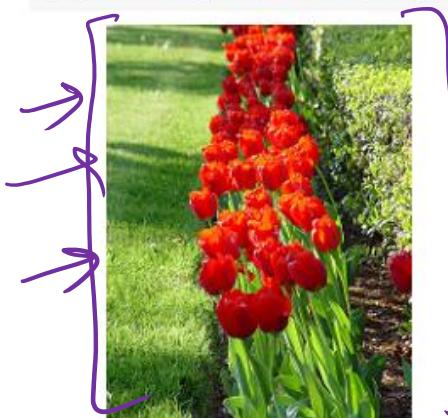


```
[ ]
```

```
PIL.Image.open(str(roses[1]))
```



```
[ ] PIL.Image.open(str(tulips[1]))
```



Define some parameters for the loader:

```
[ ] batch_size = 32  
      img_height = 180  
      img_width = 180
```

"samples together" than "resize"

# of samples taken together to run / update back-propagation (update) iteration.

$y = \text{sum}(a, b, c);$

$y = \text{sum}(a, b, c)$

$y = \text{sum}($

a,

b,

c

```
[ ] train_ds = tf.keras.utils.image_dataset_from_directory  
      data_dir,  
      validation_split=0.2,  
      subset="training",  
      seed=123,  
      image_size=(img_height, img_width),  
      batch_size=batch_size)
```

randomizing seed

range [0, 1]

pre-processing

Found 3670 files belonging to 5 classes.  
Using 2936 files for training.

at flowers.  
20% Val ~ 80% Train

of entire dataset  
 $0.2 \approx 20\%$  → amount of data to be used for validation.  
 $80\% \rightarrow$  training.

```
[ ] val_ds = tf.keras.utils.image_dataset_from_directory(  
    data_dir,  
    validation_split=0.2,  
    subset="validation",  
    seed=123,  
    image_size=(img_height, img_width),  
    batch_size=batch_size)
```

Found 3670 files belonging to 5 classes.  
Using 734 files for validation.

You can find the class names in the `class_names` attribute on these datasets. These correspond to the directory names in alphabetical order.

```
[ ] class_names = train_ds.class_names  
print(class_names)
```

How to find?  
just folder names.

('daisy', 'dandelion', 'roses', 'sunflowers', 'tulips')

## ▼ Visualize the data

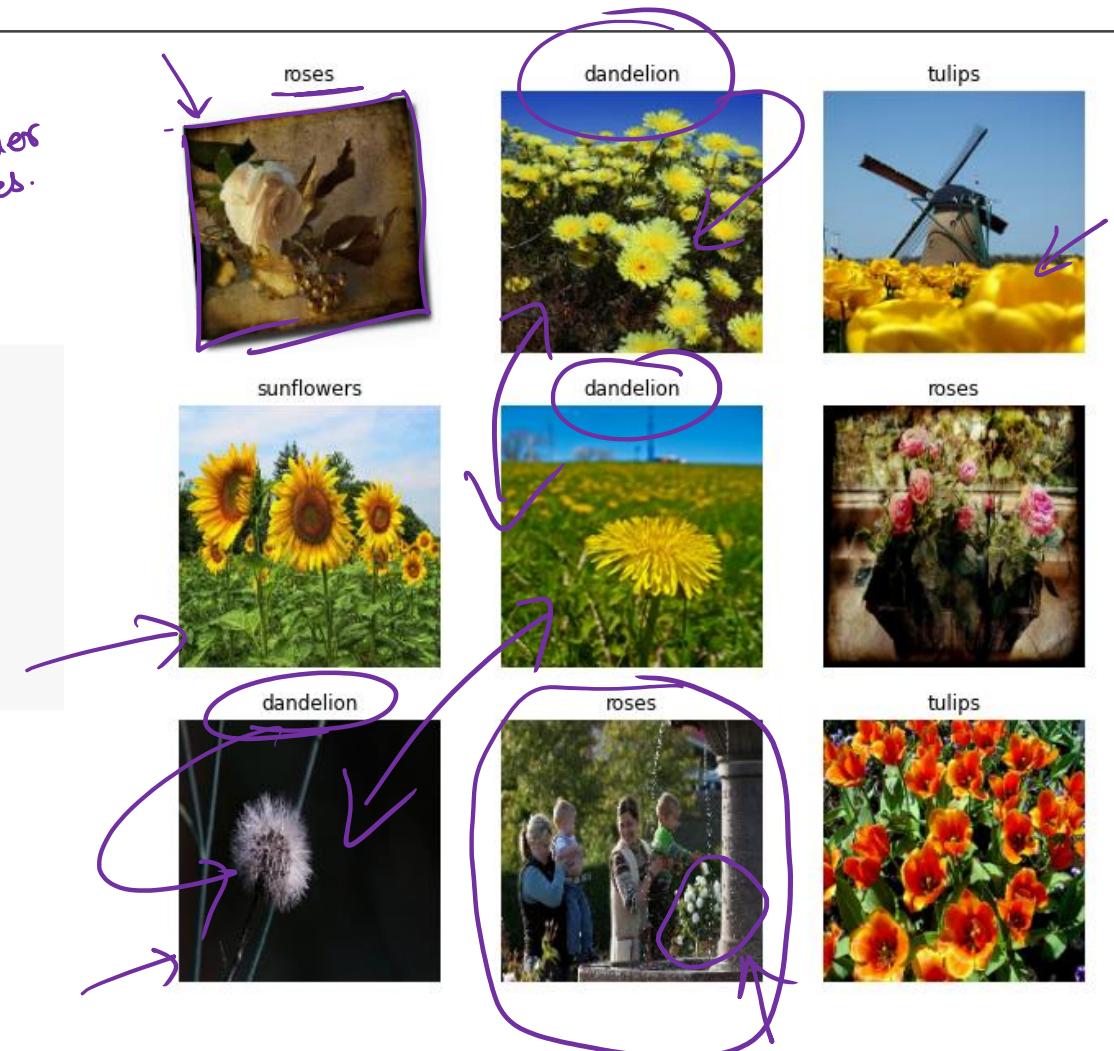
Here are the first nine images from the training dataset:

```
[ ] import matplotlib.pyplot as plt  
  
plt.figure(figsize=(10, 10))  
for images, labels in train_ds.take(1):  
    for i in range(9):  
        ax = plt.subplot(3, 3, i + 1)  
        plt.imshow(images[i].numpy().astype("uint8"))  
        plt.title(class_names[labels[i]])  
        plt.axis("off")
```



unsigned int format

ground truths.  
→ subfolder names.



You will pass these datasets to the Keras `Model.fit` method for training later in this tutorial. If you like, you can also manually iterate over the dataset and retrieve batches of images:

*We will create*

```
[ ] for image_batch, labels_batch in train_ds:  
    print(image_batch.shape) *  
    print(labels_batch.shape)  
    break
```

*32 Samples* → *each sample is image* → *RGB*  $180 \times 180 \times 3$

*(32, 180, 180, 3)* → *Batch size, H, W, C* → *channels.*

## Configure the dataset for performance

Make sure to use buffered prefetching, so you can yield data from disk without having I/O become blocking.

```
AUTOTUNE = tf.data.AUTOTUNE  
  
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)  
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

## ▼ Standardize the data

The RGB channel values are in the  $[0, 255]$  range. This is not ideal for a neural network; in general you should seek to make your input values small.

Here, you will standardize values to be in the  $[0, 1]$  range by using `tf.keras.layers.Rescaling`:

```
[ ] normalization_layer = layers.Rescaling(1./255)
```

*(image inputs → floating variable)*

$\frac{1}{255} \times \text{each pixel}$

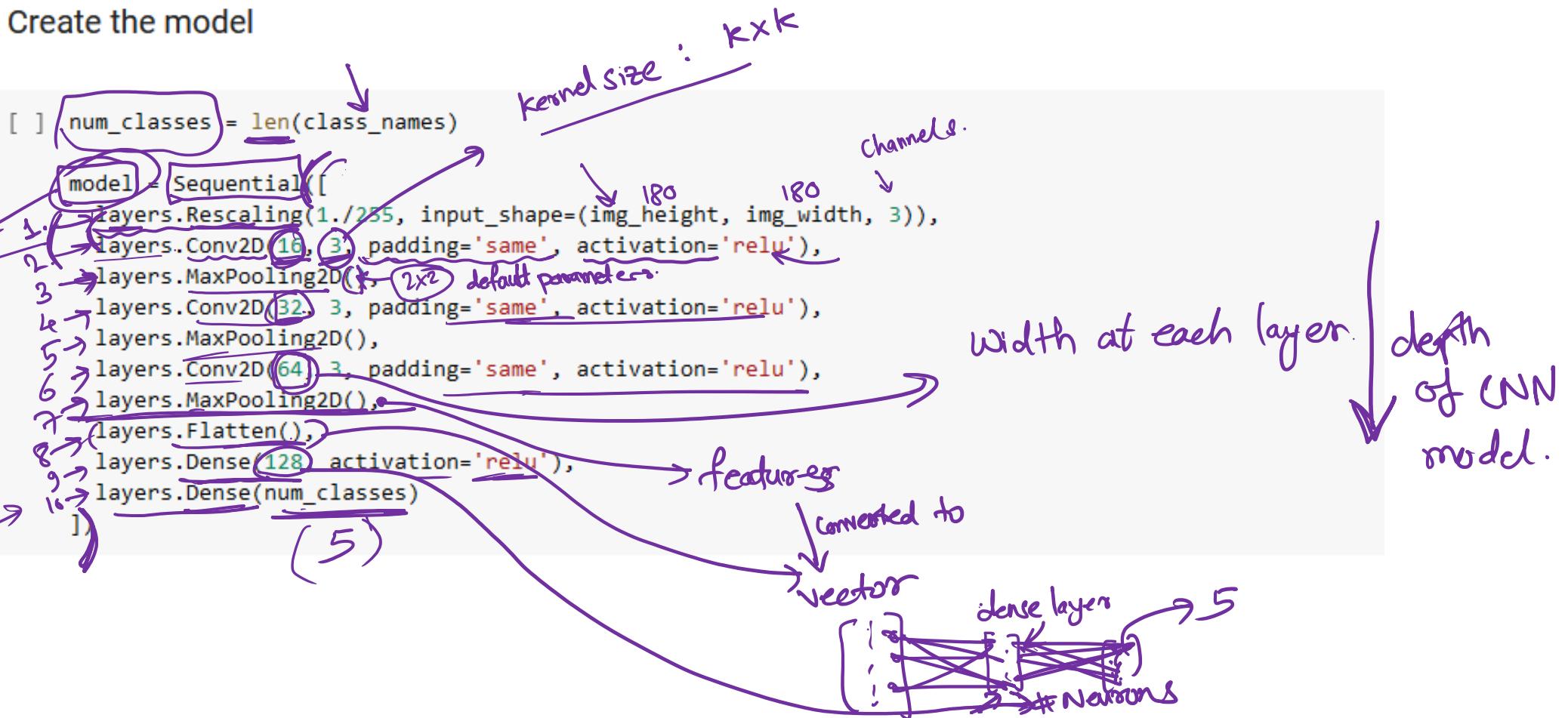
There are two ways to use this layer. You can apply it to the dataset by calling `Dataset.map`:

```
[ ] normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
# Notice the pixel values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image))
```

*0.0 1.0*

## ▼ A basic CNN model

Create the model



## ▼ Compile the model

SGD → stochastic gradient descent.

```
[ ] model.compile(optimizer='adam',
      loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
      metrics=['accuracy'])
```

### ▼ Model summary

View all the layers of the network using the Keras `Model.summary` method:

```
[ ] model.summary()
```

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
rescaling_2 (Rescaling)	(None, 180, 180, 3)	0
conv2d_3 (Conv2D)	(None, 180, 180, 16)	448
max_pooling2d_3 (MaxPooling 2D)	(None, 90, 90, 16)	0
conv2d_4 (Conv2D)	(None, 90, 90, 32)	4640
max_pooling2d_4 (MaxPooling 2D)	(None, 45, 45, 32)	0
conv2d_5 (Conv2D)	(None, 45, 45, 64)	18496
max_pooling2d_5 (MaxPooling 2D)	(None, 22, 22, 64)	0
flatten_1 (Flatten)	(None, 30976)	0
dense_2 (Dense)	(None, 128)	3965056
dense_3 (Dense)	(None, 5)	645
Total params 3,989,285		4 million parameters.
Trainable params: 3,989,285		
Non-trainable params: 0		

$$\begin{aligned} & \text{kernel } 3 \times 3 \times 3 = (27 + 1 \text{ bias}) \times 16 \\ & \frac{28}{16} \\ & 448 \end{aligned}$$

## ▼ Train the model

Train the model for 10 epochs with the Keras Model.fit method:

```
[ ] epochs=10  
history = model.fit(  
    train_ds,  
    validation_data=val_ds,  
    epochs=epochs  
)
```

If we train enough then we can always get 100% training accuracy but if validation accuracy is not good then it is "overtraining"

Training Validation

```
Epoch 1/10  
92/92 [=====] - 11s 37ms/step - loss: 1.3572 - accuracy: 0.4012 - val_loss: 1.1089 - val_accuracy: 0.5681  
Epoch 2/10  
92/92 [=====] - 2s 23ms/step - loss: 1.0010 - accuracy: 0.6046 - val_loss: 0.9536 - val_accuracy: 0.5940  
Epoch 3/10  
92/92 [=====] - 2s 23ms/step - loss: 0.8055 - accuracy: 0.7027 - val_loss: 0.9053 - val_accuracy: 0.6594  
Epoch 4/10  
92/92 [=====] - 2s 24ms/step - loss: 0.5934 - accuracy: 0.7854 - val_loss: 0.9686 - val_accuracy: 0.6526  
Epoch 5/10  
92/92 [=====] - 2s 24ms/step - loss: 0.3918 - accuracy: 0.8658 - val_loss: 1.0422 - val_accuracy: 0.6594  
Epoch 6/10  
92/92 [=====] - 2s 25ms/step - loss: 0.2247 - accuracy: 0.9244 - val_loss: 1.1349 - val_accuracy: 0.6594  
Epoch 7/10  
92/92 [=====] - 2s 23ms/step - loss: 0.1493 - accuracy: 0.9554 - val_loss: 1.3704 - val_accuracy: 0.6608  
Epoch 8/10  
92/92 [=====] - 2s 23ms/step - loss: 0.1043 - accuracy: 0.9687 - val_loss: 1.5848 - val_accuracy: 0.6730  
Epoch 9/10  
92/92 [=====] - 2s 23ms/step - loss: 0.0328 - accuracy: 0.9939 - val_loss: 1.9913 - val_accuracy: 0.6485  
Epoch 10/10  
92/92 [=====] - 2s 25ms/step - loss: 0.0170 - accuracy: 0.9973 - val_loss: 2.1011 - val_accuracy: 0.6553
```

3rd / 4th epoch  
10th epoch

## ▼ Visualize training results

Create plots of the loss and accuracy on the training and validation sets:

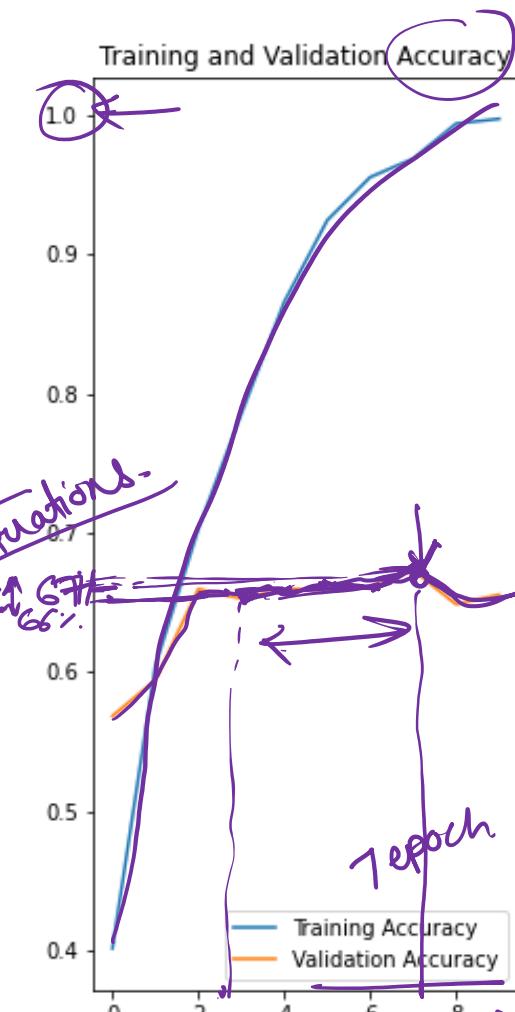
```
[ ] acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']

    loss = history.history['loss']
    val_loss = history.history['val_loss']

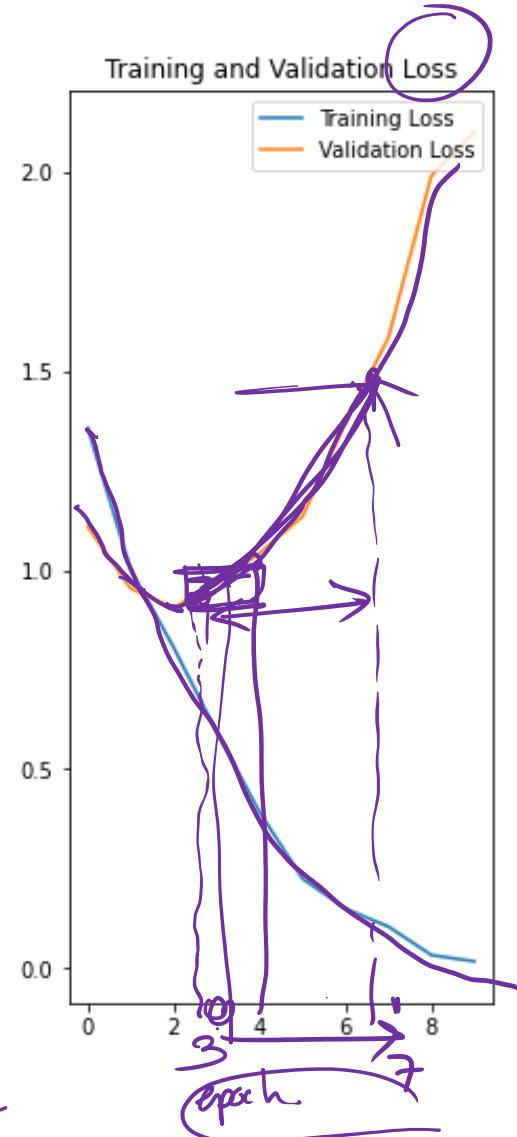
    epochs_range = range(epochs)

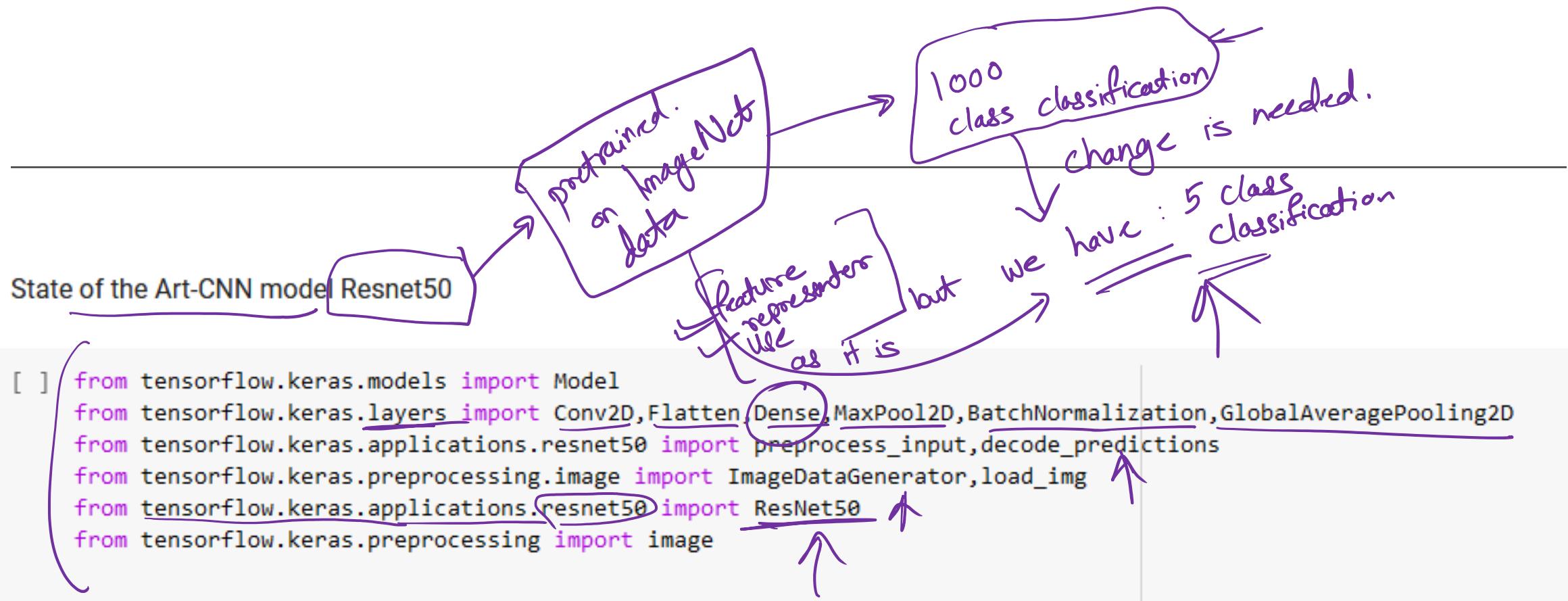
    plt.figure(figsize=(8, 8))
    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, acc, label='Training Accuracy')
    plt.plot(epochs_range, val_acc, label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.title('Training and Validation Accuracy')

    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, loss, label='Training Loss')
    plt.plot(epochs_range, val_loss, label='Validation Loss')
    plt.legend(loc='upper right')
    plt.title('Training and Validation Loss')
    plt.show()
```



stop training  
at 3rd or 4th epoch





```
[ ] base_model = ResNet50(include_top=False,weights= 'imagenet')
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024,activation='relu')(x)
predictions = Dense(5,activation='softmax')(x)
model = Model(inputs=base_model.input,outputs=predictions)

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

model.compile(optimizer='adam',loss='categorical_crossentropy',metrics = ['accuracy'])

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5
94765736/94765736 [=====] - 0s 0us/step
```

ResNet50

*pretrained*

```
[ ] model.compile(optimizer='adam',
                  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  metrics=['accuracy'])
```

## Model summary

View all the layers of the network using the Keras Model.summary method:

[ ] model.summary()			
conv2_block3_1_bn (BatchNormalization)	(None, None, None, 64)	256	['conv2_block3_1_conv[0][0]']
conv2_block3_1_relu (Activation)	(None, None, None, 64)	0	['conv2_block3_1_bn[0][0]']
conv2_block3_2_conv (Conv2D)	(None, None, None, 64)	36928	['conv2_block3_1_relu[0][0]']
conv2_block3_2_bn (BatchNormalization)	(None, None, None, 64)	256	['conv2_block3_2_conv[0][0]']
conv2_block3_2_relu (Activation)	(None, None, None, 64)	0	['conv2_block3_2_bn[0][0]']
conv2_block3_3_conv (Conv2D)	(None, None, None, 256)	16640	['conv2_block3_2_relu[0][0]']
conv2_block3_3_bn (BatchNormalization)	(None, None, None, 256)	1024	['conv2_block3_3_conv[0][0]']
.			
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0	['conv5_block3_out[0][0]']
dense_4 (Dense)	(None, 1024)	2098176	['global_average_pooling2d[0][0]']
dense_5 (Dense)	(None, 5)	5125	['dense_4[0][0]']
-----			
Total params:	25,691,013		
Trainable params:	2,103,301		
Non-trainable params:	23,587,712		

ResNet50

25 million

```
epochs=10  
history = model.fit(  
    train_ds,  
    validation_data=val_ds,  
    epochs=epochs  
)
```

```
Epoch 1/10  
/usr/local/lib/python3.8/dist-packages/tensorflow/python/util/dispatch.py:1087: UserWarning: ``sparse_categorical_crossentropy` received  
    return dispatch_target(*args, **kwargs)  
92/92 [=====] - 14s 110ms/step - loss: 0.7490 - accuracy: 0.7793 - val_loss: 0.3433 - val_accuracy: 0.8828  
Epoch 2/10  
92/92 [=====] - 8s 89ms/step - loss: 0.2634 - accuracy: 0.9077 - val_loss: 0.3493 - val_accuracy: 0.8815  
Epoch 3/10  
92/92 [=====] - 8s 88ms/step - loss: 0.1460 - accuracy: 0.9482 - val_loss: 0.4676 - val_accuracy: 0.8447  
Epoch 4/10  
92/92 [=====] - 8s 89ms/step - loss: 0.0860 - accuracy: 0.9741 - val_loss: 0.4188 - val_accuracy: 0.8719  
Epoch 5/10  
92/92 [=====] - 8s 90ms/step - loss: 0.0444 - accuracy: 0.9905 - val_loss: 0.3737 - val_accuracy: 0.8815  
Epoch 6/10  
92/92 [=====] - 8s 91ms/step - loss: 0.0207 - accuracy: 0.9966 - val_loss: 0.4846 - val_accuracy: 0.8583  
Epoch 7/10  
92/92 [=====] - 9s 102ms/step - loss: 0.0175 - accuracy: 0.9976 - val_loss: 0.3954 - val_accuracy: 0.9005  
Epoch 8/10  
92/92 [=====] - 8s 91ms/step - loss: 0.0056 - accuracy: 1.0000 - val_loss: 0.4361 - val_accuracy: 0.8828  
Epoch 9/10  
92/92 [=====] - 8s 91ms/step - loss: 0.0034 - accuracy: 1.0000 - val_loss: 0.4233 - val_accuracy: 0.8951  
Epoch 10/10  
92/92 [=====] - 8s 91ms/step - loss: 0.0019 - accuracy: 1.0000 - val_loss: 0.4222 - val_accuracy: 0.8910
```

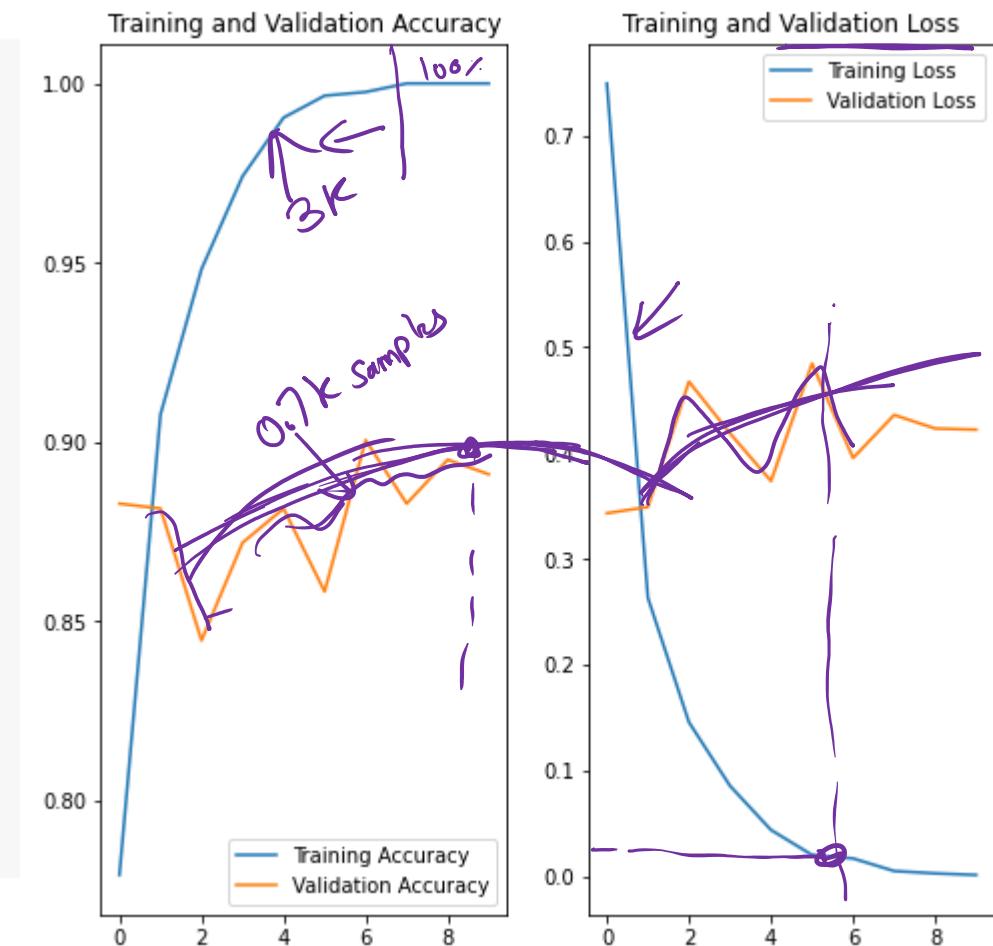
```
[ ] acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



---

Thank you