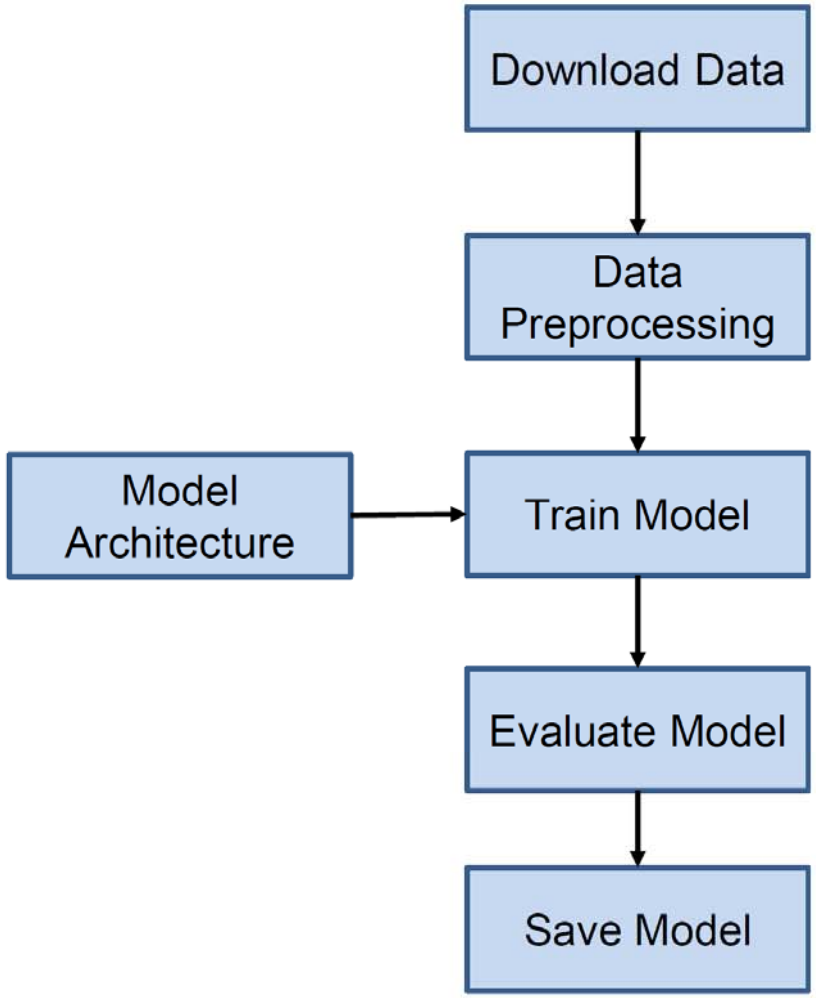
Lab 1
Cat & Dog classification and CNN model experiment

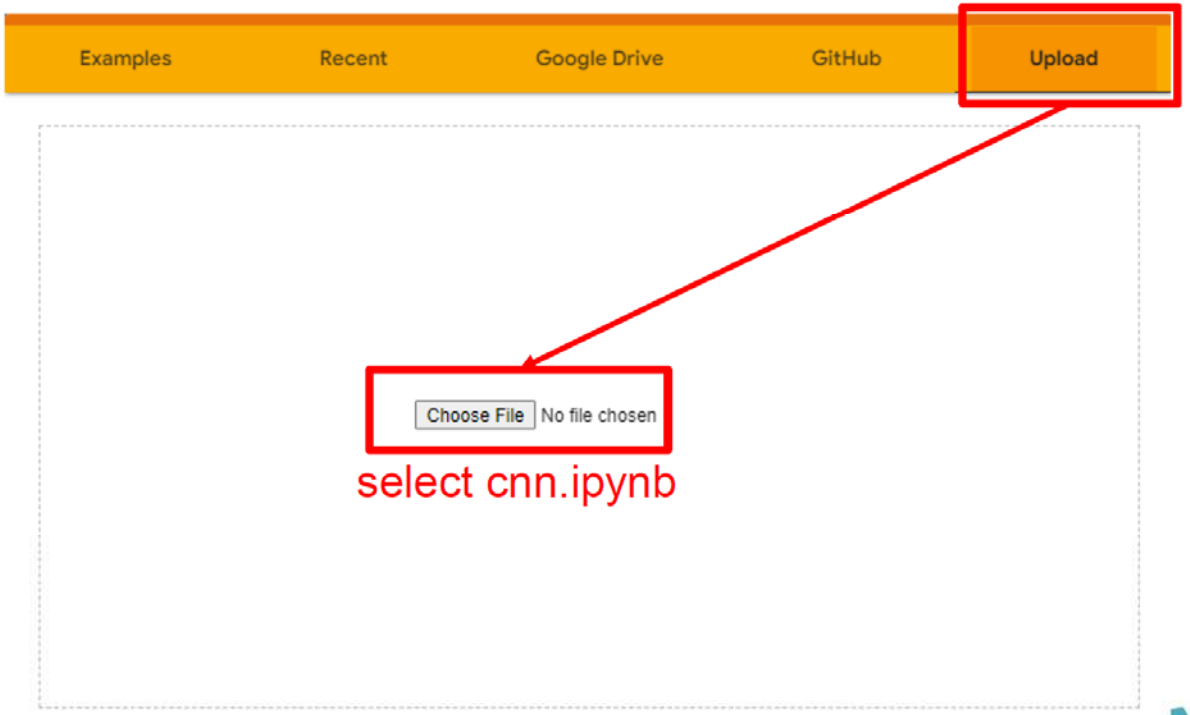
Flowchart


Download Data
Data Preprocessing
Model Architecture
Train Model
Evaluate Model
Save Model


BEST
Biomedical & Embedded System Technology Group

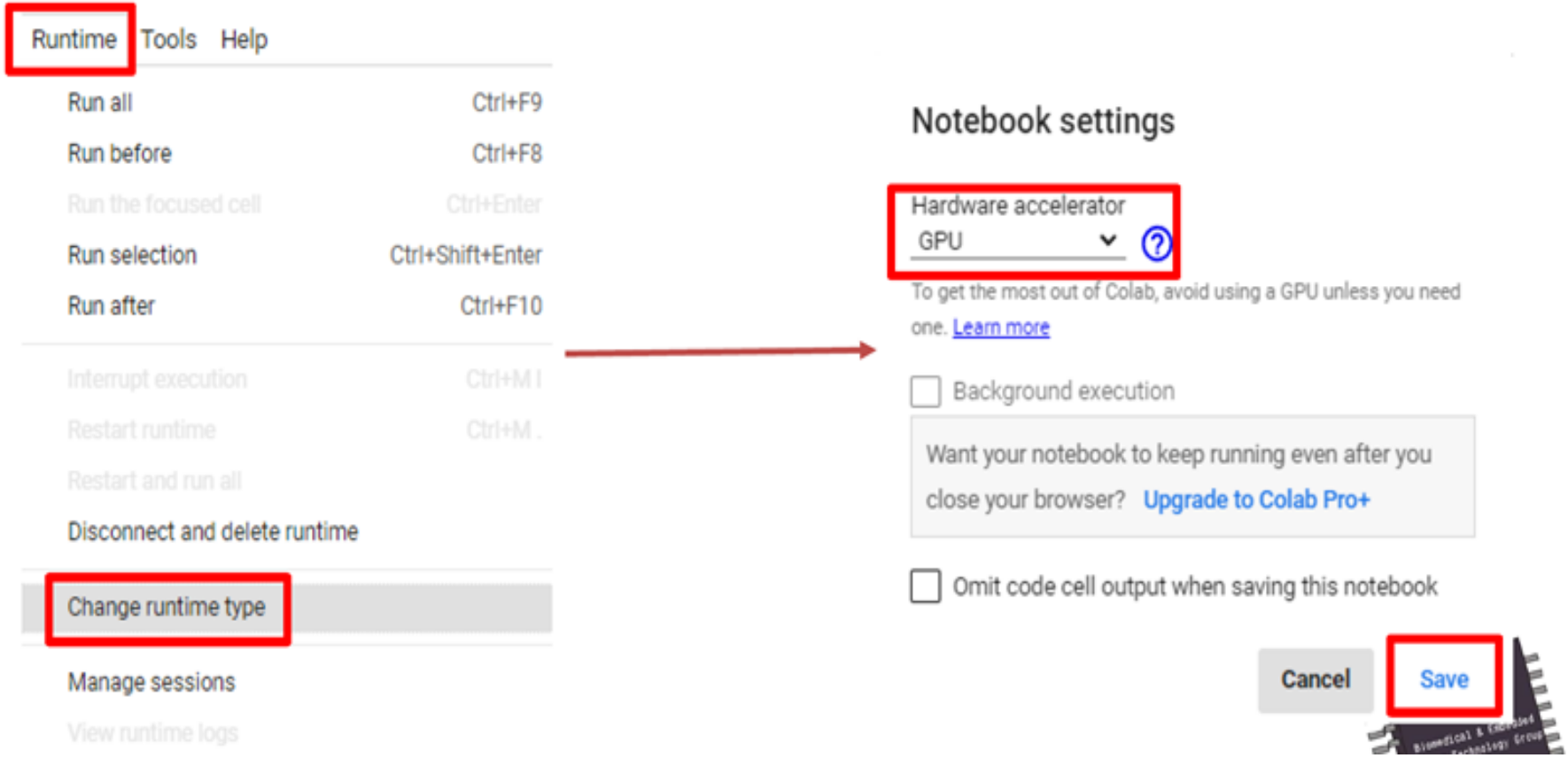
Open Colab

https://colab.research.google.com/?utm_source=scs-index



Colab setting

- Computing on a GPU in Colab.



Download dataset

- The dataset is excerpted from the "Dogs vs. Cats" dataset on Kaggle.

- Train data : 2000 JPG of cats and dogs
- Validation data : 1000 JPG of cats and dogs

```
!wget --no-check-certificate \
    https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip \
    -O /tmp/cats_and_dogs_filtered.zip
```

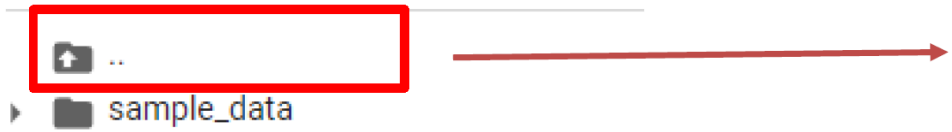
Download dataset

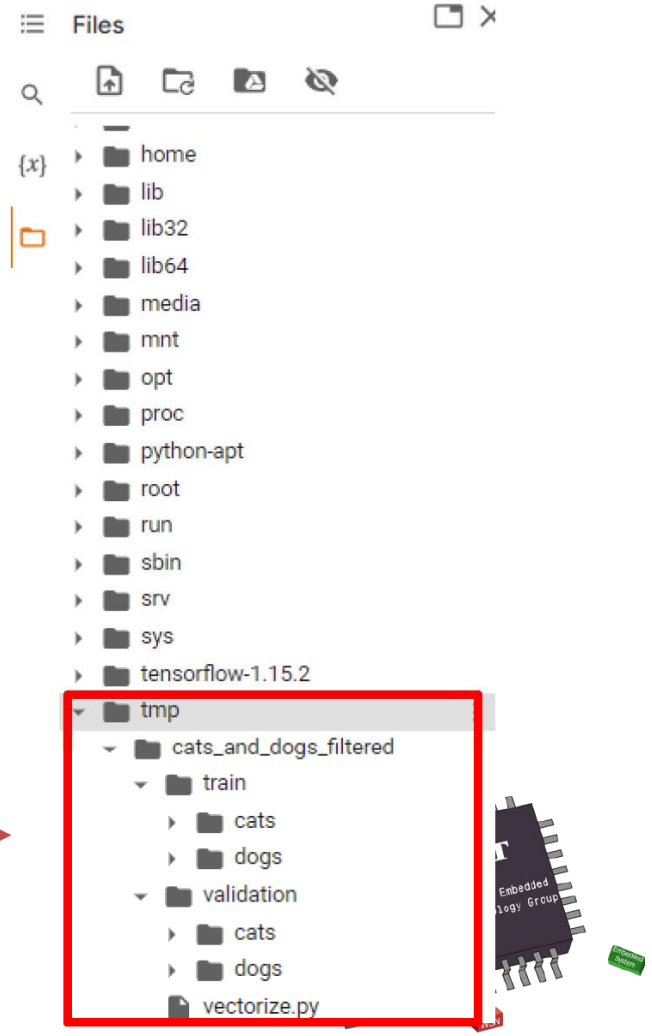
- Unzip the file.

```
import  zipfile

local_zip  =  '/tmp/cats_and_dogs_filtered.zip'
zip_ref  =  zipfile.ZipFile(local_zip,  'r')
zip_ref.extractall('/tmp')
zip_ref.close()
```

Download dataset

- Check the file hierarchy.
  - Go to the parent directory and check tmp/cats_and_dogs_filtered.





Data preprocessing

- Use image_dataset_from_directory to load image.

```
image_size = (224, 224)
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
        "/tmp/cats_and_dogs_filtered/train",
        seed=1337,
        image_size=image_size,
        label_mode="binary",
        batch_size=20
)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
        "/tmp/cats_and_dogs_filtered/validation",
        seed=1337,
        image_size=image_size,
        label_mode="binary",
        batch_size=20
)
```

Data preprocessing

- Seed : random seed for shuffling data.
- Label_mode : ‘binary’ means that the labels are encoded as scalars with values 0 or 1.
- Image_size : resize the image.

```
image_size  =  (224,224)
train_ds  =  tf.keras.preprocessing.image_dataset_from_directory(
        "/tmp/cats_and_dogs_filtered/train",
        seed=1337,
        image_size=image_size,
        label_mode="binary",
        batch_size=20
)
```

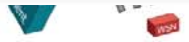
Data preprocessing

- Rescale an input in the [0, 255] range to be in the [0, 1] range.

```
normalization_layer = tf.keras.layers.Rescaling(1./255)
train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
val_ds = val_ds.map(lambda x, y: (normalization_layer(x), y))
```

- Check the value.

```
image_batch, labels_batch = next(iter(train_ds))
first_image = image_batch[0]
print(np.min(first_image), np.max(first_image))
```



Show image with label

```
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()*255).astype("uint8"))
                plt.title(int(labels[i]))
                plt.axis("off")
```

0 : cat
1 : dog

Model architecture

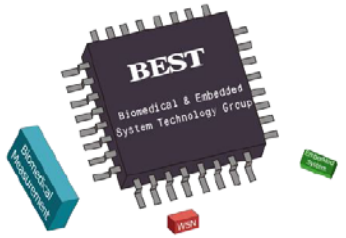
- Import the layers you need.

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense
from tensorflow.keras.layers import Activation, Flatten, Input
```

- Define input layer.

```
input_tensor=Input(shape=(224,224,3))
```

The shape of image



Model architecture

- Forward propagation the value by cascade layer.

```
input_tensor=Input(shape=(224,224,3))

x  =  Conv2D(64,(3,3),activation='relu',padding='same')(input_tensor)
x  =  Conv2D(32,(3,3),activation='relu',padding='same')(x)
x  =  MaxPooling2D((2,2),strides=(2,2))(x)

x  =  Conv2D(16,(3,3),activation='relu',padding='same')(x)
x  =  Conv2D(16,(3,3),activation='relu',padding='same')(x)
x  =  MaxPooling2D((2,2),strides=(2,2))(x)

x  =  Flatten()(x)
x  =  Dense(128,activation='relu')(x)
output_tensor  =  Dense(1,activation='sigmoid')(x)
model  =  Model(input_tensor,output_tensor)
```

Model architecture

```
x = Conv2D(16, (3, 3), activation='relu', padding='same')(x)
```

Number of kernel

Kernel size

One of “valid” or “same”.
“valid” means no padding.
“same” padding with zero.

Commonly used ‘relu’, ‘sigmoid’, ‘softmax’, ‘tanh’

Model architecture

```
x = MaxPooling2D((2, 2), strides=(2, 2))(x)
```

Pool size

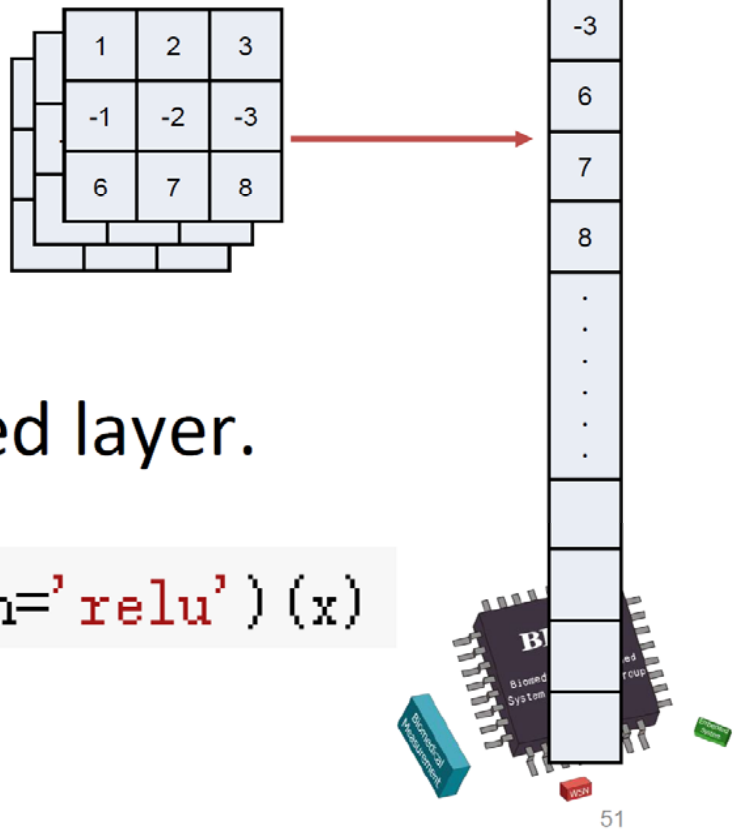
Specifies how far the pooling window moves for each pooling step.

50

Model architecture

- Change the 3d array to 1d array.

```
x = Flatten()(x)
```



- Dense means fully connected layer.

```
x = Dense(128, activation='relu')(x)
```

Number of neuron

Check your model

```
model.summary()
```

Layer (type)	Output Shape	Param #
input_11 (InputLayer)	[(None, 224, 224, 3)]	0
conv2d_25 (Conv2D)	(None, 224, 224, 64)	1792
conv2d_26 (Conv2D)	(None, 224, 224, 32)	18464
max_pooling2d_11 (MaxPooling2D)	(None, 112, 112, 32)	0
conv2d_27 (Conv2D)	(None, 112, 112, 16)	4624
conv2d_28 (Conv2D)	(None, 112, 112, 16)	2320
max_pooling2d_12 (MaxPooling2D)	(None, 56, 56, 16)	0
flatten_9 (Flatten)	(None, 50176)	0
dense_18 (Dense)	(None, 128)	6422656
dense_19 (Dense)	(None, 1)	129

Total params: 6,449,985
Trainable params: 6,449,985
Non-trainable params: 0

Train model

- Configure the model, set optimizer and loss function.

```
adam = tf.keras.optimizers.Adam(learning_rate=0.001)
momentum = tf.keras.optimizers.SGD(learning_rate=0.001, momentum=0.9)

model.compile(optimizer=momentum, loss='binary_crossentropy', metrics=['accuracy'])
```

Now is "momentum"

53

Train model

- Train the model and store record of training loss values and metrics values.

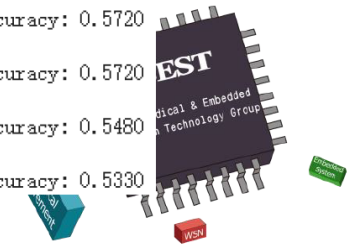
- Epochs : Number of epochs to train the model.

```
hist  =  model.fit(train_ds,
                   epochs=100,
                   validation_data=val_ds,
                   verbose=1)
```

54

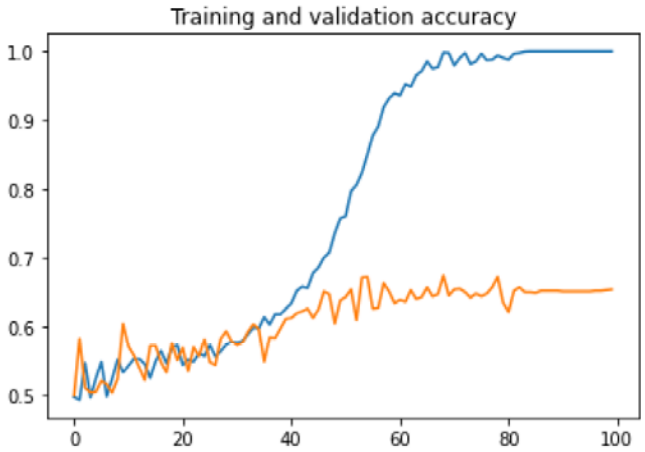
Train model

```
Epoch 1/100
100/100 [==============================] - 111s 709ms/step - loss: 0.6932 - accuracy: 0.4975 - val_loss: 0.6931 - val_accuracy: 0.5000
Epoch 2/100
100/100 [==============================] - 71s 707ms/step - loss: 0.6931 - accuracy: 0.4935 - val_loss: 0.6930 - val_accuracy: 0.5830
Epoch 3/100
100/100 [==============================] - 71s 708ms/step - loss: 0.6930 - accuracy: 0.5470 - val_loss: 0.6929 - val_accuracy: 0.5110
Epoch 4/100
100/100 [==============================] - 71s 707ms/step - loss: 0.6931 - accuracy: 0.4970 - val_loss: 0.6929 - val_accuracy: 0.5040
Epoch 5/100
100/100 [==============================] - 71s 708ms/step - loss: 0.6929 - accuracy: 0.5245 - val_loss: 0.6928 - val_accuracy: 0.5050
Epoch 6/100
100/100 [==============================] - 71s 707ms/step - loss: 0.6928 - accuracy: 0.5485 - val_loss: 0.6927 - val_accuracy: 0.5210
Epoch 7/100
100/100 [==============================] - 71s 708ms/step - loss: 0.6929 - accuracy: 0.4985 - val_loss: 0.6926 - val_accuracy: 0.5160
Epoch 8/100
100/100 [==============================] - 71s 708ms/step - loss: 0.6928 - accuracy: 0.5240 - val_loss: 0.6926 - val_accuracy: 0.5040
Epoch 9/100
100/100 [==============================] - 71s 708ms/step - loss: 0.6926 - accuracy: 0.5520 - val_loss: 0.6924 - val_accuracy: 0.5240
Epoch 10/100
100/100 [==============================] - 71s 707ms/step - loss: 0.6926 - accuracy: 0.5330 - val_loss: 0.6923 - val_accuracy: 0.6050
Epoch 11/100
100/100 [==============================] - 71s 708ms/step - loss: 0.6926 - accuracy: 0.5430 - val_loss: 0.6922 - val_accuracy: 0.5710
Epoch 12/100
100/100 [==============================] - 71s 707ms/step - loss: 0.6923 - accuracy: 0.5530 - val_loss: 0.6920 - val_accuracy: 0.5580
Epoch 13/100
100/100 [==============================] - 71s 708ms/step - loss: 0.6921 - accuracy: 0.5530 - val_loss: 0.6918 - val_accuracy: 0.5400
Epoch 14/100
100/100 [==============================] - 71s 707ms/step - loss: 0.6921 - accuracy: 0.5455 - val_loss: 0.6917 - val_accuracy: 0.5220
Epoch 15/100
100/100 [==============================] - 71s 708ms/step - loss: 0.6920 - accuracy: 0.5250 - val_loss: 0.6913 - val_accuracy: 0.5720
Epoch 16/100
100/100 [==============================] - 71s 708ms/step - loss: 0.6916 - accuracy: 0.5495 - val_loss: 0.6910 - val_accuracy: 0.5720
Epoch 17/100
100/100 [==============================] - 71s 708ms/step - loss: 0.6911 - accuracy: 0.5645 - val_loss: 0.6907 - val_accuracy: 0.5480
Epoch 18/100
100/100 [==============================] - 71s 708ms/step - loss: 0.6911 - accuracy: 0.5465 - val_loss: 0.6905 - val_accuracy: 0.5330
```



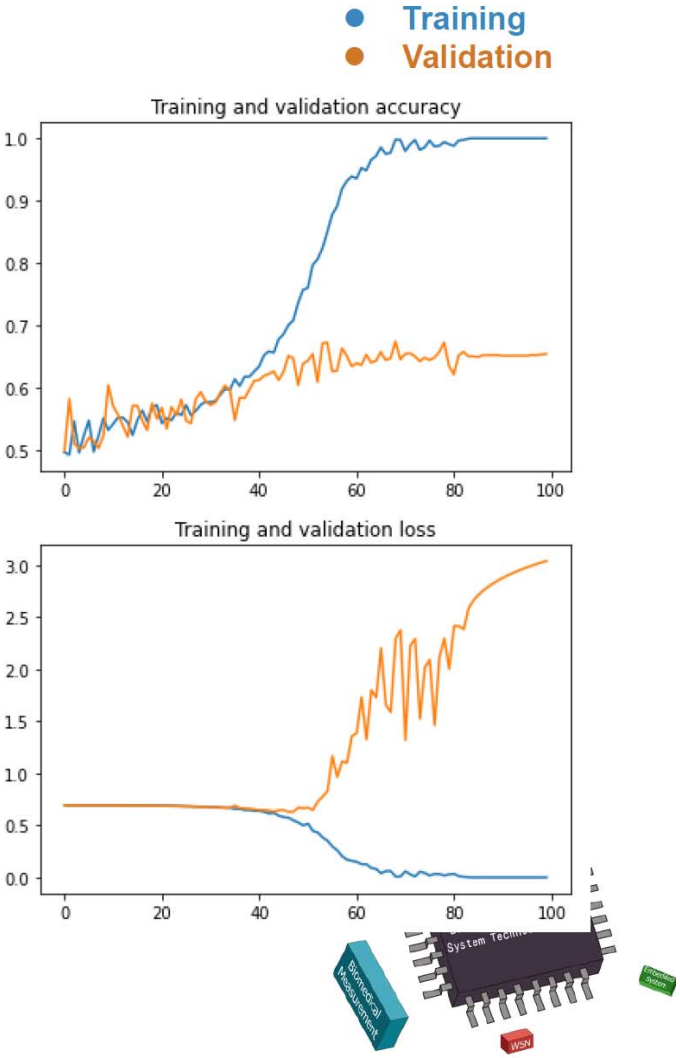
Evaluate model

```
import matplotlib.pyplot as plt
from keras.preprocessing.image import img_to_array, load_img

# Retrieve a list of accuracy results on training and test data
# sets for each training epoch
acc = hist.history['accuracy']
val_acc = hist.history['val_accuracy']

# Retrieve a list of list results on training and test data
# sets for each training epoch
loss = hist.history['loss']
val_loss = hist.history['val_loss']

# Get number of epochs
epochs = range(len(acc))

# Plot training and validation accuracy per epoch
plt.plot(epochs, acc)
plt.plot(epochs, val_acc)
plt.title('Training and validation accuracy')

plt.figure()

# Plot training and validation loss per epoch
plt.plot(epochs, loss)
plt.plot(epochs, val_loss)
plt.title('Training and validation loss')
```



Evaluate model

- As you can see, the model is overfitting.
- Our training accuracy gets close to 100%, but validation accuracy stalls at 60%.



Overfitting

- Overfitting is a tough problem when we train the model.
- Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data.

Save / load model

- Save your model to deploy on different platform.

- You can find your file in /content/myModel.h5.

```
model.save("myModel.h5")

new_model = tf.keras.models.load_model('myModel.h5')

new_model.evaluate(val_ds, verbose=1)
```

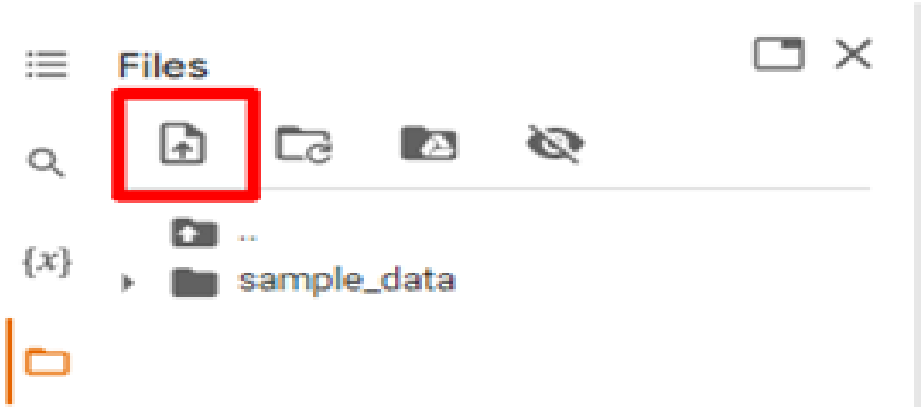
```
50/50 [==============================] - 11s 214ms/step - loss: 0.0730 - accuracy: 0.9770
[0.07302631437778473, 0.9769999980926514]
```

```
animal = "cat" # cat or dog
index = 2222 # 2000-2499
img_path = f"/tmp/cats_and_dogs_filtered/validation/{animal}s/{animal}.{index}.jpg"
img = load_img(img_path, target_size=(224, 224))  # this is a PIL image
plt.title(img_path)
plt.axis("off")
plt.imshow(img)

x = img_to_array(img)  # Numpy array with shape (224, 224, 3)
x = x.reshape((1,) + x.shape)  # Numpy array with shape (1, 224, 224, 3)
x /= 255 # Rescale by 1/255

import time
start = time.time() # record start time

result = new_model.predict(x) # inference(predict)
if result < 0.5:
  print("It is a cat.")
else:
  print("It is a dog.")
finish = time.time() # record finish time

print ("Result = %f" %result)
print("Test time :%f second." %(finish-start))
```

```
It is a cat.
Result = 0.004889
Test time :0.167567 second.
/tmp/cats_and_dogs_filtered/validation/cats/cat.2222.jpg
```

Upload/load model

- Upload pre-trained model and evaluate the model.




```
model  =  tf.keras.models.load_model("myModel.h5")

model.evaluate(val_ds,verbose=1)
```

```
50/50 [==============================] - 494s 10s/step - loss: 0.0730 - accuracy: 0.9770
[0.07302630692720413, 0.9769999980926514]
```

Assignment 1

- Select two of the models in the right chart.
- Implement and compare their accuracies.

A	B	C	D	E
conv3-4	conv3-4 conv3-4	conv3-4 conv3-4	conv3-4 conv3-4	conv3-4 conv3-4
maxpool(size=(2,2),strides=(2,2))				
conv3-8	conv3-8 conv3-8	conv3-8 conv3-8	conv3-8 conv3-8	conv3-8 conv3-8
maxpool(size=(2,2),strides=(2,2))				
conv3-16 conv3-16	conv3-16 conv3-16	conv3-16 conv3-16 conv3-16	conv3-16 conv3-16 conv3-16 conv3-16	conv3-16 conv3-16 conv3-16 conv3-16
maxpool(size=(2,2),strides=(2,2))				
conv3-32 conv3-32	conv3-32 conv3-32	conv3-32 conv3-32 conv3-32	conv3-32 conv3-32 conv3-32	conv3-32 conv3-32 conv3-32 conv3-32
maxpool(size=(2,2),strides=(2,2))				
conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64 conv3-64	conv3-64 conv3-64 conv3-64	conv3-64 conv3-64 conv3-64 conv3-64
maxpool(size=(2,2),strides=(2,2))				
FC-128				
FC-64				
FC-1				
sigmoid				

ConvA-B => kernel size: (A,A), number of the kernerl: B
FC-N => number of neuron: N

Assignment 1

- Finish a report.
  - What is observed & learned?
  - Screen dump:
    - Model.summary()
    - Plot training and validation accuracy
    - Plot training and validation loss

Submission

- Hierarchy:
  - Model_1.ipynb
  - Model_2.ipynb
  - Report.pdf

- Compress all above files in a single zip file named StudentID_lab1.zip.

References

- K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv:1409.1556, 2014.
- Stanford University CS231n, "Deep Learning for Computer Vision." [Online]. Available: http://cs231n.stanford.edu/.
- H. Y. Lee, "Hung-yi Lee youtube channel" [Online]. Available: https://www.youtube.com/c/HungyiLeeNTU.
- Kaggle, "Dogs vs. Cats." [Online]. Available: https://www.kaggle.com/c/dogs-vs-cats.