```
from google.colab import drive
drive.mount('/content/drive')
```

```
     Mounted at /content/drive
```

```
train_path = "/content/drive/MyDrive/best_model/train"
```

```
test_path = "/content/drive/MyDrive/best_model/test"
```

```
from imutils import paths
import random
SEED=10

imagePaths_train = sorted(list(paths.list_images(train_path)))
random.seed(SEED)
random.shuffle(imagePaths_train)
imagePaths_train[:5]
```

```
     ['/content/drive/MyDrive/best_model/train/cat.27.jpg',
      '/content/drive/MyDrive/best_model/train/cat.145.jpg',
      '/content/drive/MyDrive/best_model/train/dog.12390.jpg',
      '/content/drive/MyDrive/best_model/train/cat.121.jpg',
      '/content/drive/MyDrive/best_model/train/dog.12356.jpg']
```

```
import cv2
from tensorflow.keras.preprocessing.image import img_to_array

image = cv2.imread(imagePaths_train[0])
print("Shape of image = ", image.shape)

image = cv2.resize(image,(28,28))
print("Shape of resize image = ", image.shape)

image = img_to_array(image)

image
```

```
     Shape of image =  (479, 370, 3)
     Shape of resize image =  (28, 28, 3)
     array([[[160., 157., 149.],
             [116., 111., 108.],
             [100.,  98.,  94.],
             ...,
             [ 75.,  85.,  85.],
             [ 78.,  88.,  88.],
             [ 85.,  95.,  95.]],

            [[143., 140., 132.],
             [102., 101.,  97.],
             [ 91.,  90.,  86.],
             ...,
             [ 72.,  82.,  82.],
             [ 73.,  83.,  83.],
             [ 77.,  91.,  90.]],

            [[145., 143., 135.],
             [111., 110., 106.],
             [ 85.,  84.,  80.],
             ...,
             [ 69.,  79.,  79.],
             [ 74.,  84.,  84.],
             [ 75.,  87.,  87.]],

            ...,

            [[139., 145., 180.],
             [126., 134., 170.],
             [109., 117., 164.],
             ...,
             [106., 113., 162.],
             [103., 110., 159.],
             [105., 111., 161.]],

            [[142., 149., 188.],
             [115., 123., 170.],
             [120., 128., 175.],
             ...,
             [105., 115., 170.],
             [ 90., 102., 150.],
             [ 94., 102., 142.]],

            [[123., 130., 181.],
             [116., 129., 174.],
             [117., 127., 181.],
             ...,
             [133., 141., 201.],
             [ 90.,  96., 161.],
             [106., 117., 174.]]], dtype=float32)
```

```
import os
imagePaths_train[0].split("/")[-1].split(".")[0]
```

```
'cat'
```

```
from tqdm import tqdm_notebook as tqdm

# initialize the data and labels
print("[INFO] loading images...")
train_X = []
train_Y = []

# grab the image paths and randomly shuffle them
imagePaths = sorted(list(paths.list_images(train_path)))
random.seed(SEED)
random.shuffle(imagePaths)

# progress bar
with tqdm(total=len(imagePaths)) as pbar:
    # loop over the input images
    for idx, imagePath in enumerate(imagePaths):
        # load the image, pre-process it, and store it in the data list
        image = cv2.imread(imagePath)
        image = cv2.resize(image, (28, 28))
        image = img_to_array(image)
        train_X.append(image)

        # extract the class label from the image path and update the
        # labels list
        label = imagePath.split(os.path.sep)[-1].split(".")[0]

        if label == "cat":
            label = 0
        elif label == "dog":
            label = 1

        # print("pr: ", label)

        train_Y.append(label)

        # update the progressbar
        pbar.update(1)
```

```
[INFO] loading images...
<ipython-input-127-75ac8f014d3a>:14: TqdmDeprecationWarning: This function
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  with tqdm(total=len(imagePaths)) as pbar:
100%                                            396/396 [00:02<00:00, 217.23it/s]
```

```
import numpy as np
train_X = np.array(train_X, dtype="float") / 255.0
train_Y = np.array(train_Y)

from sklearn.model_selection import train_test_split
(trainX, valX, trainY, valY) = train_test_split(train_X, train_Y, test_size=0.40, random_state=SEED)
```

```
trainX.shape
```

```
(237, 28, 28, 3)
```

```
valX.shape
```

```
(159, 28, 28, 3)
```

```
valY
```

```
array([1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1,
       0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1,
       0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1,
       0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1])
```

```
from tensorflow.keras.utils import to_categorical
trainY = to_categorical(trainY, num_classes=2)
valY = to_categorical(valY, num_classes=2)
```

```
valY
```

```
array([[0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
```

```
trainY
```

```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [1., 0.],
       [1., 0.],
           [0., 1.],
           [0., 1.],
           [0., 1.],
           [1., 0.],
           [0., 1.],
           [1., 0.],
           [1., 0.],
           [1., 0.],
           [1., 0.],
           [0., 1.],
           [1., 0.],
           [0., 1.],
           [1., 0.],
           [1., 0.],
           [0., 1.],
           [0., 1.],
           [0., 1.],
           [1., 0.],
           [1., 0.],
           [0., 1.],
           [1., 0.],
           [0., 1.],
           [0., 1.],
           [0., 1.],
           [0., 1.],
           [0., 1.],
           [0., 1.],
           [0., 1.],
           [0., 1.],
           [0., 1.],
           [1., 0.],
           [1., 0.],
           [0., 1.],
           [0., 1.],
           [1., 0.],
           [1., 0.],
           [1., 0.],
           [1., 0.],
           [1., 0.],
           [1., 0.],
           [0., 1.],
           [1., 0.]
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

aug = ImageDataGenerator(rotation_range=30,
                         width_shift_range=0.1,
                         height_shift_range=0.1,
                         shear_range=0.2,
                         zoom_range=0.2,
                         horizontal_flip=True,
                         fill_mode="nearest")
```

```
EPOCHS = 10
INIT_LR = 1e-3
BS = 2
```

```
from tensorflow.keras.layers import Dense
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation
from tensorflow.keras.layers import MaxPooling2D
```

```
# create CNN Model

class LeNet:
    @staticmethod
    def build(width, height, depth, classes):
        # initialize the model
        model = Sequential()                ## siamese networks
        inputShape = (height, width, depth)

        # if we are using "channels first", update the input shape
        print(K.image_data_format())
        if K.image_data_format() == "channels_first":
            inputShape = (depth, height, width)

        # first set of CONV => RELU => POOL layers
        model.add(Conv2D(20, (5, 5), padding="same",input_shape=inputShape))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
        # second set of CONV => RELU => POOL layers
        model.add(Conv2D(50, (5, 5), padding="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

        # first (and only) set of FC => RELU layers
        model.add(Flatten())
        model.add(Dense(500))
        model.add(Activation("relu"))

        # softmax classifier
        model.add(Dense(classes))
        model.add(Activation("softmax"))

        # return the constructed network architecture
        return model


# initialize the model
print("[INFO] compiling model...")
model = LeNet.build(width=28, height=28, depth=3, classes=2)
opt = Adam(learning_rate=INIT_LR)
model.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])
print("[INFO] model complied...")
```

```
[INFO] compiling model...
channels_last
[INFO] model complied...
```

```
print(model.summary())
```

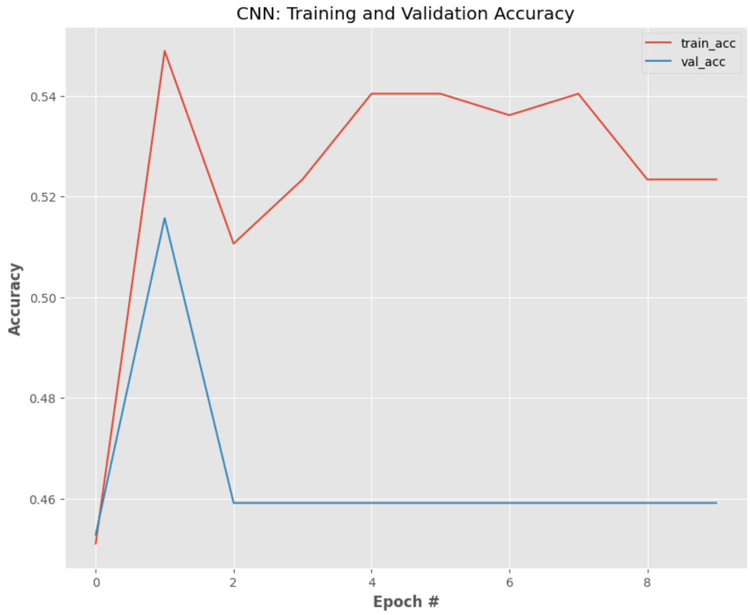
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 28, 28, 20)        1520

 activation_12 (Activation)  (None, 28, 28, 20)        0

 max_pooling2d_6 (MaxPooling  (None, 14, 14, 20)       0
 2D)

 conv2d_7 (Conv2D)           (None, 14, 14, 50)        25050

 activation_13 (Activation)  (None, 14, 14, 50)        0

 max_pooling2d_7 (MaxPooling  (None, 7, 7, 50)         0
 2D)

 flatten_3 (Flatten)         (None, 2450)              0

 dense_6 (Dense)             (None, 500)               1225500

 activation_14 (Activation)  (None, 500)               0

 dense_7 (Dense)             (None, 2)                 1002

 activation_15 (Activation)  (None, 2)                 0

=================================================================
Total params: 1,253,072
Trainable params: 1,253,072
Non-trainable params: 0
_________________________________________________________________
None
```

```
BS = 2

print("[INFO] training network...")
H = model.fit(x=aug.flow(trainX, trainY, batch_size=BS),
              validation_data=(valX, valY),
              steps_per_epoch=len(trainX) // BS,
              epochs=EPOCHS,
              verbose=1)
```

```
[INFO] training network...
Epoch 1/10
118/118 [==============================] - 4s 23ms/step - loss: 0.7085 - accuracy: 0.4511 - val_loss: 0.6952 - val_accuracy: 0.4528
Epoch 2/10
118/118 [==============================] - 3s 26ms/step - loss: 0.7040 - accuracy: 0.5489 - val_loss: 0.6911 - val_accuracy: 0.5157
Epoch 3/10
118/118 [==============================] - 3s 22ms/step - loss: 0.6950 - accuracy: 0.5106 - val_loss: 0.6975 - val_accuracy: 0.4591
Epoch 4/10
118/118 [==============================] - 4s 30ms/step - loss: 0.7091 - accuracy: 0.5234 - val_loss: 0.6937 - val_accuracy: 0.4591
```

```
Epoch 5/10
118/118 [==============================] - 3s 27ms/step - loss: 0.6921 - accuracy: 0.5404 - val_loss: 0.6993 - val_accuracy: 0.4591
Epoch 6/10
118/118 [==============================] - 3s 22ms/step - loss: 0.6918 - accuracy: 0.5404 - val_loss: 0.6958 - val_accuracy: 0.4591
Epoch 7/10
118/118 [==============================] - 3s 28ms/step - loss: 0.6922 - accuracy: 0.5362 - val_loss: 0.6994 - val_accuracy: 0.4591
Epoch 8/10
118/118 [==============================] - 3s 22ms/step - loss: 0.6896 - accuracy: 0.5404 - val_loss: 0.7063 - val_accuracy: 0.4591
Epoch 9/10
118/118 [==============================] - 3s 22ms/step - loss: 0.6841 - accuracy: 0.5234 - val_loss: 0.6986 - val_accuracy: 0.4591
Epoch 10/10
118/118 [==============================] - 3s 25ms/step - loss: 0.6957 - accuracy: 0.5234 - val_loss: 0.6980 - val_accuracy: 0.4591
```

```
# plot the training and validation accuracy
import matplotlib.pyplot as plt
N = np.arange(0, EPOCHS)
plt.style.use("ggplot")
plt.figure(figsize = [10,8])
plt.plot(N, H.history["accuracy"], label="train_acc")
plt.plot(N, H.history["val_accuracy"], label="val_acc")
plt.title("CNN: Training and Validation Accuracy")
plt.xlabel("Epoch #", weight="bold")
plt.ylabel("Accuracy", weight="bold")
plt.legend()
plt.show()
```



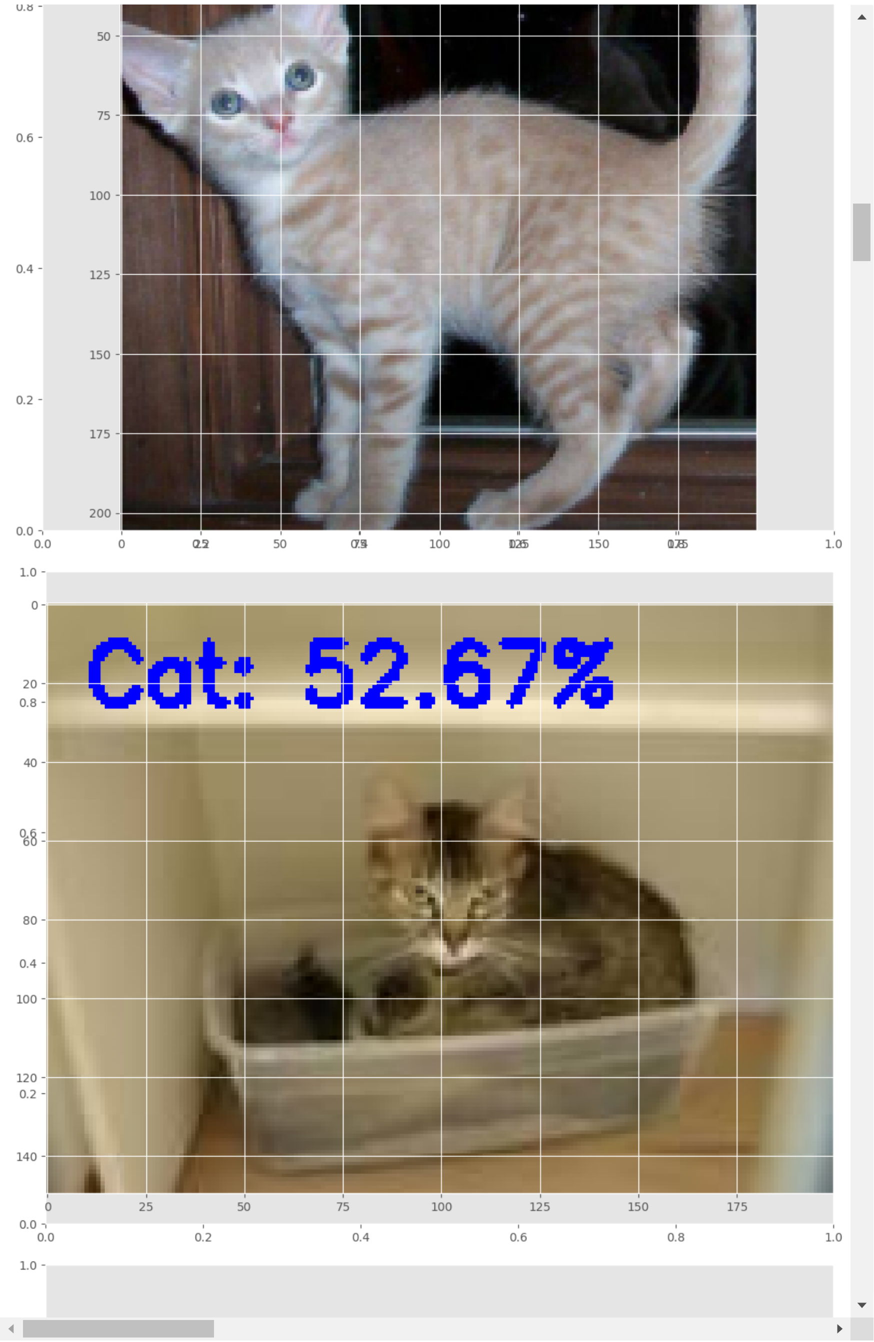
```
model.save("/content/drive/MyDrive/best_model/cat_dog_new.model")
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _update_step_xla while sa
```

```
from tensorflow.keras.models import load_model
model = load_model("/content/drive/MyDrive/best_model/cat_dog_new.model")
```

```
import imutils
def display_img(img):
    fig = plt.figure(figsize=(12,10))
    plt.grid()
    ax = fig.add_subplot(111)
    ax.imshow(img)
```

```
from tqdm import tqdm_notebook as tqdm

# initialize the data and labels
print("[INFO] loading images...")

predicted_label = []
image_numbers = []

# grab the image paths and randomly shuffle them
imagePaths = sorted(list(paths.list_images(test_path)))
random.seed(SEED)
random.shuffle(imagePaths)

# progress bar
with tqdm(total=len(imagePaths)) as pbar:
    # loop over the input images
    for idx, imagePath in enumerate(imagePaths):
        # load the image, pre-process it, and store it in the data list
        image = cv2.imread(imagePath)
        orig = image.copy()
        image = cv2.resize(image, (28, 28))
        image = image.astype("float") / 255.0
        image = img_to_array(image)
        image = np.expand_dims(image, axis=0)

        image_number = imagePath.split("/")[-1].split(".")[0]
        image_numbers.append(image_number)

        # classify the input image
        prd_conf = model.predict(image)[0]

        all_class = ["Cat","Dog"]
        # build the label
        label = all_class[np.argmax(prd_conf)]
        predicted_label.append(label)
        proba = prd_conf[np.argmax(prd_conf)]

        label = "{}: {:.2f}%".format(label, proba * 100)

        # draw the label on the image
        output = imutils.resize(orig, width=200)
        cv2.putText(output, label, (10, 25),  cv2.FONT_HERSHEY_SIMPLEX,
            0.7, (255, 0, 0), 2)

        # convert img to rgb format and display in notebook
        img = cv2.cvtColor(output, cv2.COLOR_BGR2RGB)
        display_img(img)

        pbar.update(1)
```


0.8
0.6
0.4
0.2
0.0
50
75
100
125
150
175
200
0.0
0
0.2
25
50
0.4
75
100
0.6
125
150
0.8
175
1.0
1.0
0
Cat: 52.67%
20
0.8
40
0.6
60
80
0.4
100
120
0.2
140
0
25
50
75
100
125
150
175
0.0
0.0
0.2
0.4
0.6
0.8
1.0
1.0