

Task 2 : Dogs vs. Cats Classification

Start coding or [generate](#) with AI.

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.cifar10.load_data()

train_images, test_images = train_images / 255.0, test_images / 255.0

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

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

datagen = ImageDataGenerator(rotation_range=40,
                             width_shift_range=0.2,
                             height_shift_range=0.2,
                             shear_range=0.2,
                             zoom_range=0.2,
                             horizontal_flip=True,
                             fill_mode='nearest')

history = model.fit(datagen.flow(train_images, train_labels, batch_size=32),
                    epochs=10,
                    validation_data=(test_images, test_labels))

test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f'\nTest accuracy: {test_acc}')

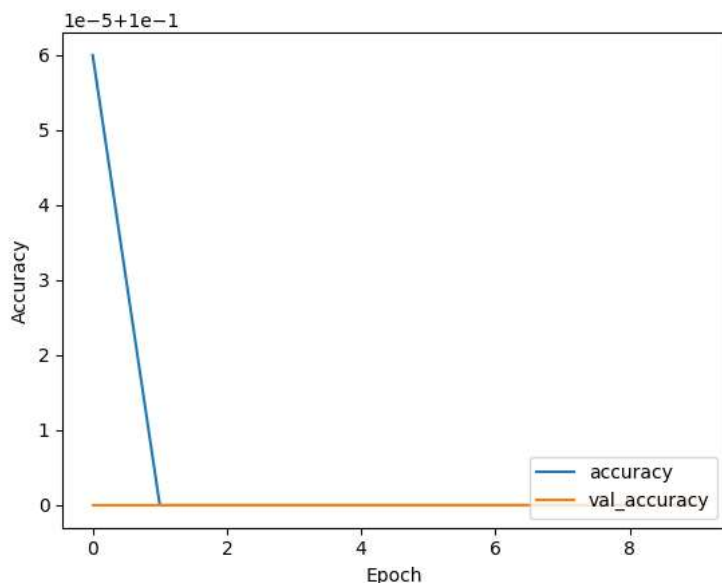
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```

```

Epoch 1/10
1563/1563 [=====] - 62s 39ms/step - loss: -11947171905536.0000 - accuracy: 0.1001 - val_loss: -69497272664064.0
Epoch 2/10
1563/1563 [=====] - 54s 34ms/step - loss: -520568558321664.0000 - accuracy: 0.1000 - val_loss: -140391367678361
Epoch 3/10
1563/1563 [=====] - 57s 36ms/step - loss: -3682190890106880.0000 - accuracy: 0.1000 - val_loss: -70976014972354
Epoch 4/10
1563/1563 [=====] - 53s 34ms/step - loss: -13261938813304832.0000 - accuracy: 0.1000 - val_loss: -2163840386701
Epoch 5/10
1563/1563 [=====] - 54s 34ms/step - loss: -34444383583469568.0000 - accuracy: 0.1000 - val_loss: -5096109263709
Epoch 6/10
1563/1563 [=====] - 54s 34ms/step - loss: -73919462360743936.0000 - accuracy: 0.1000 - val_loss: -1027561562539
Epoch 7/10
1563/1563 [=====] - 53s 34ms/step - loss: -140000274398838784.0000 - accuracy: 0.1000 - val_loss: -186073926181
Epoch 8/10
1563/1563 [=====] - 54s 35ms/step - loss: -242744215765778432.0000 - accuracy: 0.1000 - val_loss: -312463852946
Epoch 9/10
1563/1563 [=====] - 56s 36ms/step - loss: -395130325442756608.0000 - accuracy: 0.1000 - val_loss: -496242411165
Epoch 10/10
1563/1563 [=====] - 54s 35ms/step - loss: -611690479246901248.0000 - accuracy: 0.1000 - val_loss: -751923704081
313/313 - 2s - loss: -7.5192e+17 - accuracy: 0.1000 - 2s/epoch - 8ms/step

```

Test accuracy: 0.1000000149011612



```

import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.callbacks import TensorBoard

from warnings import filterwarnings
filterwarnings('ignore')

classifier = Sequential()
classifier.add(Conv2D(32,(3,3),input_shape=(64,64,3),activation = 'relu'))
classifier.add(MaxPooling2D(pool_size=(2,2),strides=2)) #if stride not given it equal to pool filter size
classifier.add(Conv2D(32,(3,3),activation = 'relu'))
classifier.add(MaxPooling2D(pool_size=(2,2),strides=2))
classifier.add(Flatten())
classifier.add(Dense(units=128,activation='relu'))
classifier.add(Dense(units=1,activation='sigmoid'))
adam = tensorflow.keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)

```

```

from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.1,
                                   zoom_range=0.1,
                                   horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)

#Training Set
train_set = train_datagen.flow_from_directory('train',
                                              target_size=(64,64),
                                              batch_size=32,
                                              class_mode='binary')

#Validation Set
test_set = test_datagen.flow_from_directory('test',
                                             target_size=(64,64),
                                             batch_size = 32,
                                             class_mode='binary',
                                             shuffle=False)

#Test Set /no output available
test_set1 = test_datagen.flow_from_directory('test1',
                                             target_size=(64,64),
                                             batch_size=32,
                                             shuffle=False)

%%capture
classifier.fit_generator(train_set,
                        steps_per_epoch=800,
                        epochs = 200,
                        validation_data = test_set,
                        validation_steps = 20,
                        #callbacks=[tensorboard]
                        );

from tensorflow.keras.models import load_model
classifier = load_model('resources/dogcat_model_bak.h5')

%matplotlib inline
import tensorflow
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt
import numpy as np
img1 = image.load_img('test/Cat/10.jpg', target_size=(64, 64))
img = image.img_to_array(img1)
img = img/255
# create a batch of size 1 [N,H,W,C]
img = np.expand_dims(img, axis=0)
prediction = classifier.predict(img, batch_size=None, steps=1) #gives all class prob.
if(prediction[:,:]>0.5):
    value = 'Dog :%1.2f'%(prediction[0,0])
    plt.text(20, 62,value,color='red', fontsize=18, bbox=dict(facecolor='white', alpha=0.8))
else:
    value = 'Cat :%1.2f'%(1.0-prediction[0,0])
    plt.text(20, 62,value,color='red', fontsize=18, bbox=dict(facecolor='white', alpha=0.8))

plt.imshow(img1)
plt.show()

plt.show('https://images.unsplash.com/photo-1608848461950-0fe51dfc41cb?q=80&w=1000&auto=format&fit=crop&ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHx1
plt.show()


![image.png](italicised text)

```



```
fig=plt.figure(figsize=(15, 6))
columns = 7
rows = 3
for i in range(columns*rows):
    fig.add_subplot(rows, columns, i+1)
    img1 = image.load_img('test1/'+test_set1 filenames[np.random.choice(range(12500))], target_size=(64, 64))
    img = image.img_to_array(img1)
    img = img/255
    img = np.expand_dims(img, axis=0)
    prediction = classifier.predict(img, batch_size=None, steps=1) #gives all class prob.
    if(prediction[:,>0.5):
        value = 'Dog :%1.2f'%(prediction[0,0])
        plt.text(20, 58,value,color='red',fontsize=10, bbox=dict(facecolor='white',alpha=0.8))
    else:
        value = 'Cat :%1.2f'%(1.0-prediction[0,0])
        plt.text(20, 58,value,color='red',fontsize=10, bbox=dict(facecolor='white',alpha=0.8))
plt.imshow(img1)
```



 image.png