Convolutional Neural Network

Importing the libraries

In [1]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
2025-06-10 11:21:02.927911: I tensorflow/core/platform/cpu_feature_guard.cc:210] This Ten
sorFlow binary is optimized to use available CPU instructions in performance-critical ope
rations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow w
ith the appropriate compiler flags.
```

In [2]:
```
tf.__version__
```

Out[2]:
```
'2.16.2'
```

Part 1 - Data Preprocessing

Preprocessing the Training set

In [5]:
```
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)
training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'binary')
```

```
Found 8000 images belonging to 2 classes.
```

Preprocessing the Test set

In [7]:
```
test_datagen = ImageDataGenerator(rescale = 1./255)
test_set = test_datagen.flow_from_directory('dataset/test_set',
                                            target_size = (64, 64),
                                            batch_size = 32,
                                            class_mode = 'binary')
```

```
Found 2000 images belonging to 2 classes.
```

Part 2 - Building the CNN

Initialising the CNN

In [9]:
```
cnn = tf.keras.models.Sequential()
```

Step 1 - Convolution

In [11]:
```
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu', input_shape
```

```
/opt/anaconda3/envs/moflow/lib/python3.10/site-packages/keras/src/layers/convolutional/ba
se_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a laye
r. When using Sequential models, prefer using an `Input(shape)` object as the first layer
in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Step 2 - Pooling

In [13]:
```
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

Adding a second convolutional layer

In [15]:
```
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

Step 3 - Flattening

In [17]:
```
cnn.add(tf.keras.layers.Flatten())
```

Step 4 - Full Connection

In [19]:
```
cnn.add(tf.keras.layers.Dense(units=128, activation='relu'))
```

Step 5 - Output Layer

In [21]:
```
cnn.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

Part 3 - Training the CNN

Compiling the CNN

In [23]:
```
cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Training the CNN on the Training set and evaluating it on the Test set

In [25]:
```
cnn.fit(x = training_set, validation_data = test_set, epochs = 25)
```

```
/opt/anaconda3/envs/moflow/lib/python3.10/site-packages/keras/src/trainers/data_adapters/
py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__ini
t__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing
`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
```

```
Epoch 1/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 45s 173ms/step - accuracy: 0.5680 - loss: 0.6761 - val_accur
acy: 0.6930 - val_loss: 0.5953
Epoch 2/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 39s 158ms/step - accuracy: 0.6733 - loss: 0.5999 - val_accur
acy: 0.7235 - val_loss: 0.5550
Epoch 3/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 38s 150ms/step - accuracy: 0.7277 - loss: 0.5475 - val_accur
acy: 0.7515 - val_loss: 0.5137
Epoch 4/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 40s 160ms/step - accuracy: 0.7456 - loss: 0.5202 - val_accur
acy: 0.7475 - val_loss: 0.5169
Epoch 5/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 38s 154ms/step - accuracy: 0.7578 - loss: 0.4912 - val_accur
acy: 0.7730 - val_loss: 0.4799
Epoch 6/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 39s 156ms/step - accuracy: 0.7652 - loss: 0.4852 - val_accur
acy: 0.7655 - val_loss: 0.4911
Epoch 7/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 39s 157ms/step - accuracy: 0.7884 - loss: 0.4475 - val_accur
acy: 0.7760 - val_loss: 0.4922
Epoch 8/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 39s 158ms/step - accuracy: 0.7920 - loss: 0.4345 - val_accur
acy: 0.7780 - val_loss: 0.4633
Epoch 9/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 43s 171ms/step - accuracy: 0.8009 - loss: 0.4322 - val_accur
acy: 0.7860 - val_loss: 0.4623
Epoch 10/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 39s 155ms/step - accuracy: 0.8110 - loss: 0.4090 - val_accur
acy: 0.7885 - val_loss: 0.4697
Epoch 11/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 39s 155ms/step - accuracy: 0.8220 - loss: 0.3902 - val_accur
acy: 0.7880 - val_loss: 0.4681
Epoch 12/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 41s 165ms/step - accuracy: 0.8257 - loss: 0.3855 - val_accur
acy: 0.7575 - val_loss: 0.5083
Epoch 13/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 44s 176ms/step - accuracy: 0.8272 - loss: 0.3813 - val_accur
acy: 0.7880 - val_loss: 0.4757
Epoch 14/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 39s 158ms/step - accuracy: 0.8455 - loss: 0.3579 - val_accur
acy: 0.7790 - val_loss: 0.4995
Epoch 15/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 39s 157ms/step - accuracy: 0.8421 - loss: 0.3467 - val_accur
acy: 0.7975 - val_loss: 0.4906
Epoch 16/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 38s 151ms/step - accuracy: 0.8497 - loss: 0.3401 - val_accur
acy: 0.7715 - val_loss: 0.4928
Epoch 17/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 34s 135ms/step - accuracy: 0.8616 - loss: 0.3160 - val_accur
acy: 0.8140 - val_loss: 0.4561
Epoch 18/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 33s 131ms/step - accuracy: 0.8663 - loss: 0.3100 - val_accur
acy: 0.7650 - val_loss: 0.5933
Epoch 19/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 37s 147ms/step - accuracy: 0.8669 - loss: 0.2924 - val_accur
acy: 0.8065 - val_loss: 0.4690
Epoch 20/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 38s 153ms/step - accuracy: 0.8784 - loss: 0.2779 - val_accur
acy: 0.7885 - val_loss: 0.5155
Epoch 21/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 41s 163ms/step - accuracy: 0.8861 - loss: 0.2677 - val_accur
```

```
acy: 0.7810 - val_loss: 0.5378
Epoch 22/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 39s 158ms/step - accuracy: 0.8937 - loss: 0.2548 - val_accur
acy: 0.8035 - val_loss: 0.5181
Epoch 23/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 38s 151ms/step - accuracy: 0.8999 - loss: 0.2413 - val_accur
acy: 0.8040 - val_loss: 0.5151
Epoch 24/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 36s 144ms/step - accuracy: 0.8974 - loss: 0.2367 - val_accur
acy: 0.7905 - val_loss: 0.5532
Epoch 25/25
250/250 ━━━━━━━━━━━━━━━━━━━━ 39s 155ms/step - accuracy: 0.9114 - loss: 0.2176 - val_accur
acy: 0.8005 - val_loss: 0.5455
```

Out[25]: <keras.src.callbacks.history.History at 0x188b9ba30>

Part 4 - Making a single prediction

In [31]:
```
import numpy as np
from tensorflow.keras.preprocessing import image
test_image = image.load_img('dataset/single_prediction/cat_or_dog_2.jpg', target_size =
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = cnn.predict(test_image)
training_set.class_indices
if result[0][0] == 1:
  prediction = 'dog'
else:
  prediction = 'cat'
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
```

In [33]:
```
print(prediction)
```

```
cat
```

In [ ]: