```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten
```

Show command palette (Ctrl+Shift+P)

```
model.add(Conv2D(64, (3, 3), activation='relu' , input_shape = (64,64,3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 62, 62, 64)	1,792
max_pooling2d (MaxPooling2D)	(None, 31, 31, 64)	0
flatten (Flatten)	(None, 61504)	0
dense (Dense)	(None, 128)	7,872,640
dense_1 (Dense)	(None, 10)	1,290

Total params: 7,875,722 (30.04 MB)
Trainable params: 7,875,722 (30.04 MB)
Non-trainable params: 0 (0.00 B)

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   rotation_range = 0.2,
                                   width_shift_range = 0.2,
                                   height_shift_range = 0.2,
                                   fill_mode = 'nearest',
                                   vertical_flip = True,
                                   horizontal_flip = True)
test_datagen = ImageDataGenerator(rescale = 1./255)
```

```
train_path = '//content/drive/MyDrive/Dog - panda/Dog - panda/train'
test_path =  '/content/drive/MyDrive/Dog - panda/Dog - panda/test'
train_generator = train_datagen.flow_from_directory(train_path,
                                                    target_size = (64, 64),
                                                    batch_size = 8,
                                                    class_mode = 'binary')
test_generator = test_datagen.flow_from_directory(test_path,
                                                    target_size = (64, 64),
                                                    batch_size = 8,
                                                    class_mode = 'binary')
```

```
Found 25 images belonging to 2 classes.
Found 6 images belonging to 2 classes.
```

```
train_generator.class_indices
```

```
{'dog train': 0, 'panda train': 1}
```

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

```
model.fit(train_generator, epochs=100, validation_data=test_generator)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` clas
  self._warn_if_super_not_called()
Epoch 1/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 7s 2s/step - accuracy: 0.5985 - loss: 6.8775 - val_accuracy: 0.5000 - val_loss: 4.9833
Epoch 2/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 5s 175ms/step - accuracy: 0.5131 - loss: 4.4604 - val_accuracy: 0.5000 - val_loss: 4.0338
Epoch 3/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 166ms/step - accuracy: 0.5330 - loss: 2.6101 - val_accuracy: 0.5000 - val_loss: 1.0837
Epoch 4/100
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 166ms/step - accuracy: 0.7521 - loss: 0.5008 - val_accuracy: 0.3333 - val_loss: 0.8712
Epoch 5/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 160ms/step - accuracy: 0.5751 - loss: 0.6420 - val_accuracy: 0.6667 - val_loss: 1.0739
Epoch 6/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 150ms/step - accuracy: 0.7880 - loss: 0.4636 - val_accuracy: 0.6667 - val_loss: 1.3181
Epoch 7/100
Show command palette (Ctrl+Shift+P) ━━ 1s 178ms/step - accuracy: 0.9340 - loss: 0.3717 - val_accuracy: 0.6667 - val_loss: 1.4952
Epoch 8/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 170ms/step - accuracy: 0.8292 - loss: 0.4239 - val_accuracy: 0.6667 - val_loss: 1.6271
Epoch 9/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 173ms/step - accuracy: 0.9138 - loss: 0.3226 - val_accuracy: 0.5000 - val_loss: 1.5360
Epoch 10/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 269ms/step - accuracy: 1.0000 - loss: 0.2947 - val_accuracy: 0.5000 - val_loss: 1.7787
Epoch 11/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 244ms/step - accuracy: 0.9632 - loss: 0.2451 - val_accuracy: 0.6667 - val_loss: 2.3523
Epoch 12/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 267ms/step - accuracy: 1.0000 - loss: 0.2171 - val_accuracy: 0.5000 - val_loss: 2.2516
Epoch 13/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 176ms/step - accuracy: 0.9070 - loss: 0.1801 - val_accuracy: 0.6667 - val_loss: 3.2986
Epoch 14/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 158ms/step - accuracy: 0.9562 - loss: 0.1460 - val_accuracy: 0.6667 - val_loss: 3.2072
Epoch 15/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 154ms/step - accuracy: 1.0000 - loss: 0.1555 - val_accuracy: 0.5000 - val_loss: 2.6174
Epoch 16/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 167ms/step - accuracy: 0.9222 - loss: 0.2040 - val_accuracy: 0.5000 - val_loss: 2.9597
Epoch 17/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 137ms/step - accuracy: 0.9382 - loss: 0.1103 - val_accuracy: 0.6667 - val_loss: 4.5538
Epoch 18/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 144ms/step - accuracy: 0.9757 - loss: 0.2089 - val_accuracy: 0.5000 - val_loss: 3.4134
Epoch 19/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 138ms/step - accuracy: 0.9347 - loss: 0.1151 - val_accuracy: 0.5000 - val_loss: 3.3794
Epoch 20/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 159ms/step - accuracy: 1.0000 - loss: 0.0824 - val_accuracy: 0.6667 - val_loss: 4.1160
Epoch 21/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 180ms/step - accuracy: 1.0000 - loss: 0.0337 - val_accuracy: 0.6667 - val_loss: 4.7898
Epoch 22/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 163ms/step - accuracy: 0.9445 - loss: 0.1182 - val_accuracy: 0.6667 - val_loss: 4.3671
Epoch 23/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 169ms/step - accuracy: 1.0000 - loss: 0.0296 - val_accuracy: 0.5000 - val_loss: 3.5280
Epoch 24/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 149ms/step - accuracy: 0.9250 - loss: 0.1330 - val_accuracy: 0.6667 - val_loss: 4.1604
Epoch 25/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 267ms/step - accuracy: 1.0000 - loss: 0.0574 - val_accuracy: 0.6667 - val_loss: 5.2138
Epoch 26/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 247ms/step - accuracy: 0.9632 - loss: 0.1040 - val_accuracy: 0.5000 - val_loss: 4.1340
Epoch 27/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 236ms/step - accuracy: 1.0000 - loss: 0.0426 - val_accuracy: 0.5000 - val_loss: 3.9567
```

```
model.save('dog-panda-classifier.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is consi
```

Start coding or generate with AI.