Classify Palm, Banana and a cat

Author: Shashank K Holla, 1Rv17EC139 shashankholla@outlook.in

This model below is a 3 class image classifier built to classify between a cat, banana and a hand. The model is derived from the popular VGG16 architecture but has been optimized to improve accuracy in thsi case.

Using the keras library with Tensorflow backend. Make sure to use the same versions to prevent problems

In [48]:
```
# Data Augmentation and classes
from keras.preprocessing.image import ImageDataGenerator

# NN Libraries
import numpy as np
import keras
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, MaxPool2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras import backend as K
from keras.optimizers import Adam

# Visualisation
from keras.preprocessing import image
from PIL import Image
import matplotlib.pyplot as plt

# Other
import math
import os
```

In [50]:
```
print("Tensorflow:" + tf.__version__)
print("Keras:" + keras.__version__)
```

```
Tensorflow:2.2.0
Keras2.3.0-tf
```

In [55]:
```
# Paths
train_data_dir = "../input/bananacathand-40n10/Dataset/train/"
validation_data_dir = "../input/bananacathand-40n10/Dataset/valid/"
test_data_dir = "../input/bananacathand-40n10/Dataset/test/"
```

```
├───train
│   ├───banana - 40
│   ├───cat - 40
│   └───hand - 40
├───valid
│   ├───banana - 10
│   ├───cat - 10
│   └───hand - 10
└───test
    ├───banana
    ├───cat
    └───hand
```

In [6]:
```
_inputSize      = (224,224)
_trainBatchSize = 32
_validBatchSize = 32

# Image Augmentation. Artifically generate different looking samples from the given s
amples

_rescaler            = 1. / 255
_shearRange          = 0.2
_zoomRange          = 0.15
_rotationRange       = 45
_widthShift          = 0.15
_heightShift         = 0.15
_isHorizontalFlip    = True
_isVerticalFlip      = True
_fillMode            = "nearest"
_epochs              = 100
_classes             = {0:'banana', 1:'cat', 2:'hand'}

_bestSaveModelName   = "best.h5"
_learningRate        = 0.0001

train_batches = ImageDataGenerator( rescale=_rescaler ,
                                    shear_range=_shearRange,
                                    zoom_range=_zoomRange,
                                    rotation_range=_rotationRange,
                                    width_shift_range=_widthShift,
                                    height_shift_range=_heightShift,
                                    horizontal_flip=_isHorizontalFlip,
                                    vertical_flip=_isVerticalFlip,
                                    fill_mode=_fillMode).flow_from_directory(
                                    train_data_dir,
                                    target_size =_inputSize,
                                    batch_size  =_trainBatchSize)
valid_batches = ImageDataGenerator( rescale=_rescaler ,
                                    shear_range=_shearRange,
                                    zoom_range=_zoomRange,
                                    rotation_range=_rotationRange,
                                    width_shift_range=_widthShift,
                                    height_shift_range=_heightShift,
                                    horizontal_flip=_isHorizontalFlip,
                                    vertical_flip=_isVerticalFlip,
                                    fill_mode=_fillMode).flow_from_directory(
                                    validation_data_dir,
                                    target_size =_inputSize,
                                    batch_size  =_trainBatchSize)
```

```
Found 120 images belonging to 3 classes.
Found 30 images belonging to 3 classes.
```

Visualisation of Data

The dataset has been taken from 4 places and judged to include a variety of photos.

1. Collection from people for hands (online data set was mostly fair hands with white background which might not work well if used to predict unprocessed images)
2. Most of the images scrapped from google images for educational purpose.
3. Kaggle Fruits Dataset - https://www.kaggle.com/moltean/fruits? (https://www.kaggle.com/moltean/fruits?) 4**. Kaggle Cats And Dogs Dataset - https://www.kaggle.com/c/dogs-vs-cats (https://www.kaggle.com/c/dogs-vs-cats)

In [52]:

```python
filesPerRow = 10



def printDataset(p):
    files = os.listdir(p)
    noOfCols = filesPerRow
    noOfRow = int(len(files) / noOfCols)

    # Generate the subplots
    fig, axs = plt.subplots(noOfRow, noOfCols)
    fig.set_size_inches(10, 10, forward=True)

    # Map each file to subplot
    for i in range(0, len(files)):
      file_name = files[i]
      image = Image.open(f'{p}/{file_name}')
      row = math.floor(i / filesPerRow)
      col = i % filesPerRow
      axs[row, col].imshow(image)
      axs[row, col].axis('off')

    plt.show()
```

Dataset is comprised of training data of 40 images of each class, and 10 images for validation of each class

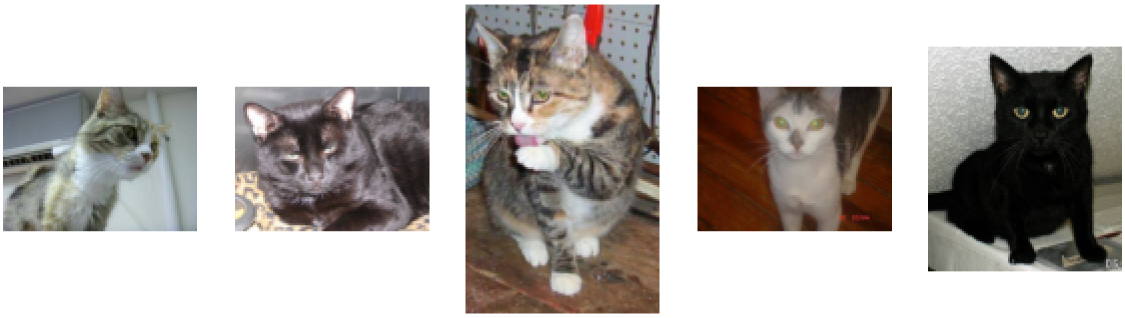
In [6]:
```
printDataset(train_data_dir+ "cat")
printDataset(train_data_dir+ "hand")
printDataset(train_data_dir+ "banana")
```

Data Augmentation is used to generate more samples from the existing set of samples by using few image processing techniques to stretch, zoom, flip and scale images. This helps the model generalise better by giving more data when dataset is limited.

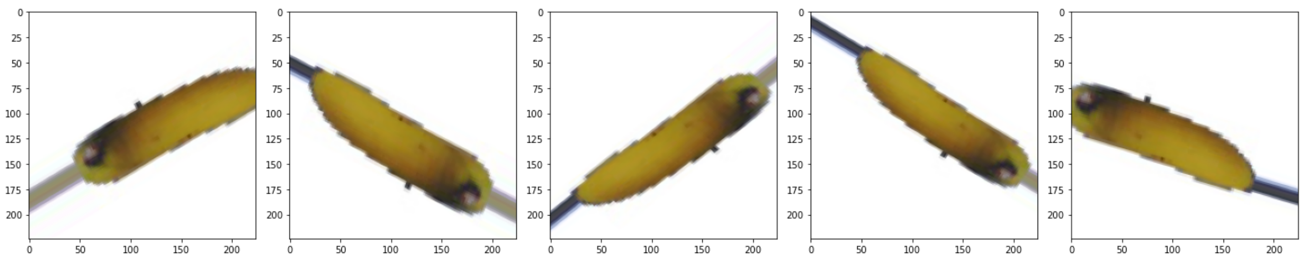
And for validation 10 images per class

In [57]:

```
filesPerRow = 5



def printDataset2(p):
    files = os.listdir(p)
    noOfCols = filesPerRow
    noOfRow = int(len(files) / noOfCols)

    # Generate the subplots
    fig, axs = plt.subplots(noOfRow, noOfCols)
    fig.set_size_inches(10, 10, forward=True)

    # Map each file to subplot
    for i in range(0, len(files)):
      file_name = files[i]
      image = Image.open(f'{p}/{file_name}')
      row = math.floor(i / filesPerRow)
      col = i % filesPerRow
      axs[row, col].imshow(image)
      axs[row, col].axis('off')

    plt.show()


printDataset2(validation_data_dir+ "cat")
printDataset2(validation_data_dir+ "hand")
printDataset2(validation_data_dir+ "banana")
```



In [7]:
```python
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 5, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip( images_arr, axes):
        ax.imshow(img)
    plt.tight_layout()
    plt.show()

augmented_images = [train_batches[0][0][1] for i in range(5)]
plotImages(augmented_images)
```



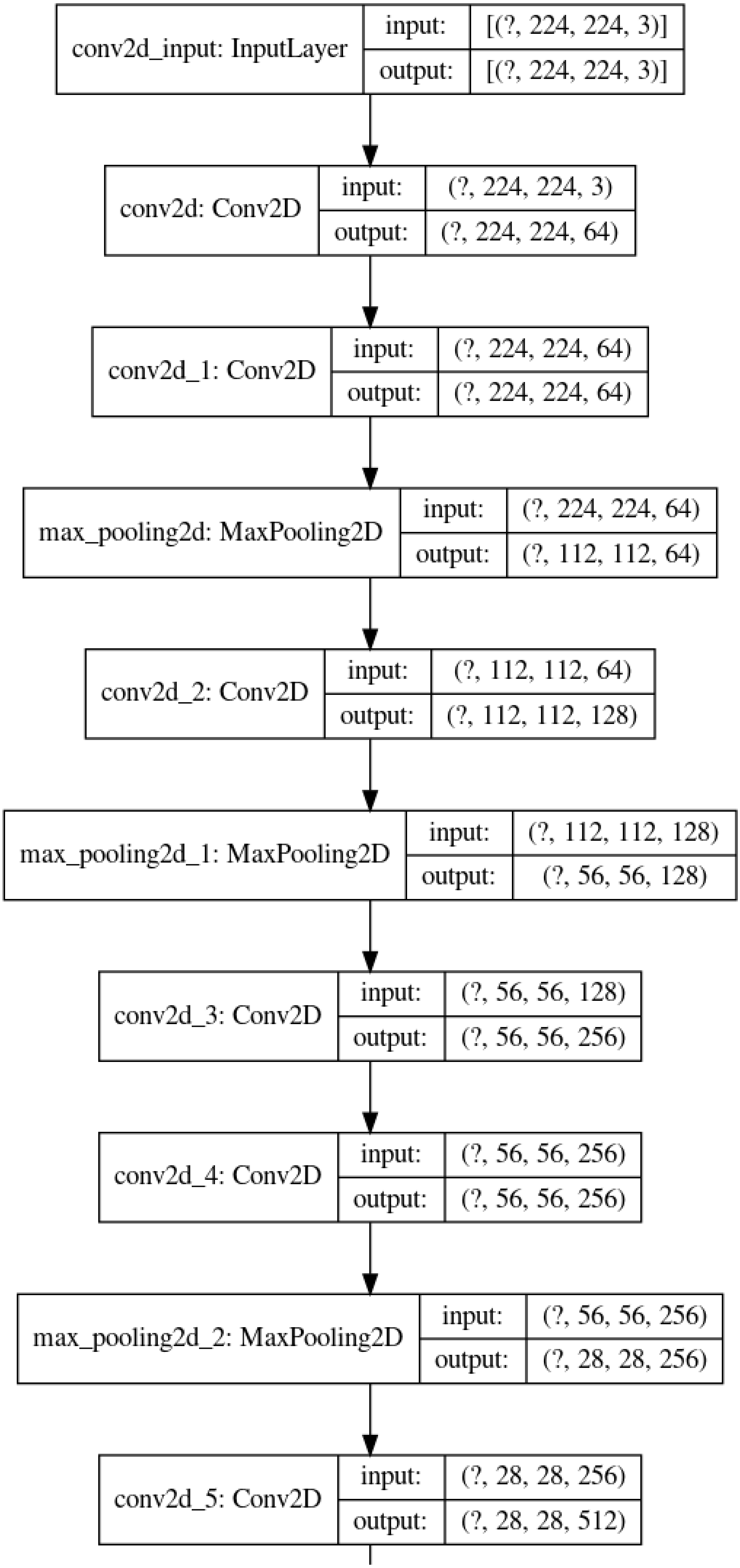
This model is a inspired from the famous VGG16 model but has been modified to reduce the layers for this application. It has an input layer of (224,224,3) with 64 output filters followed by another layer with 64 filters and "same" padding meaning the output is the same dimension as input. It is followed by a max pooling layer with stride length of (2,2). This is followed by 3 sets of similar layers with increasing number of filters to capture smaller features. Two more convolutional Dense layers with 2048 units each and ends with a 3 unit Dense layer to get the 3 classes required.

In [14]:
```
model = tf.keras.Sequential()

model.add(Conv2D(input_shape=(224,224,3),filters=64,kernel_size=(3,3),padding="same",
activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model.add(Flatten())
model.add(Dense(units=2048,activation="relu"))
model.add(Dense(units=2048,activation="relu"))
model.add(Dense(units=3, activation="softmax"))
```

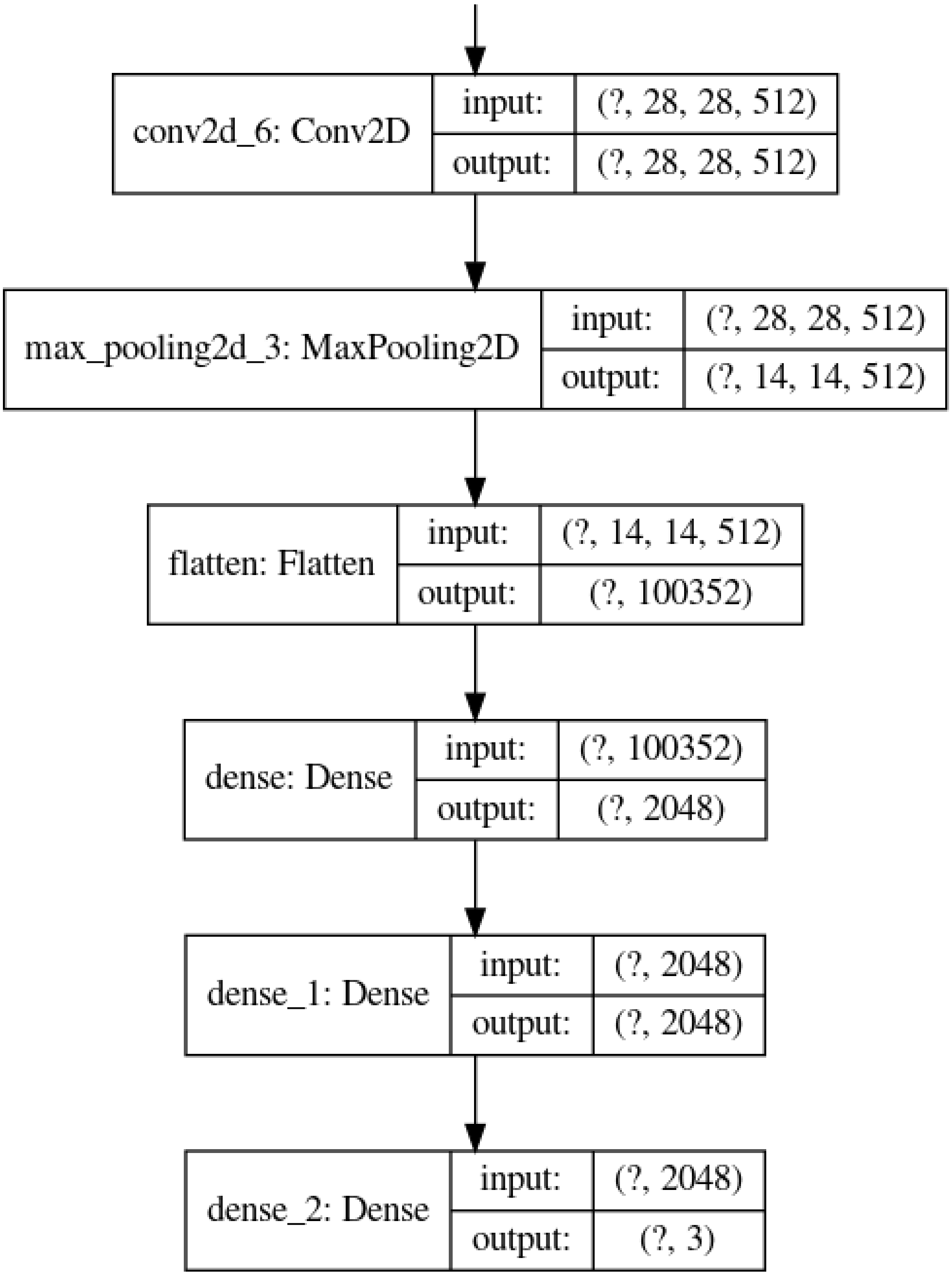
A visualization of the model.

In [10]:
```
from keras.utils.vis_utils import plot_model
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```

Out[10]:



conv2d_input: InputLayer
input: [(?, 224, 224, 3)]
output: [(?, 224, 224, 3)]
conv2d: Conv2D
input: (?, 224, 224, 3)
output: (?, 224, 224, 64)
conv2d_1: Conv2D
input: (?, 224, 224, 64)
output: (?, 224, 224, 64)
max_pooling2d: MaxPooling2D
input: (?, 224, 224, 64)
output: (?, 112, 112, 64)
conv2d_2: Conv2D
input: (?, 112, 112, 64)
output: (?, 112, 112, 128)
max_pooling2d_1: MaxPooling2D
input: (?, 112, 112, 128)
output: (?, 56, 56, 128)
conv2d_3: Conv2D
input: (?, 56, 56, 128)
output: (?, 56, 56, 256)
conv2d_4: Conv2D
input: (?, 56, 56, 256)
output: (?, 56, 56, 256)
max_pooling2d_2: MaxPooling2D
input: (?, 56, 56, 256)
output: (?, 28, 28, 256)
conv2d_5: Conv2D
input: (?, 28, 28, 256)
output: (?, 28, 28, 512)



Adam optimizer is used after very poor results after using SGD which gives erratic accuracy changes. A ModelCheckPoint call back is used to save the best model based on minimum Validation lose

In [ ]:
```
## Getting poor results with SGD. Switching back to Adam.


#opt = keras.optimizers.SGD(lr=1e-4, momentum=0.9)
#model.compile(loss="categorical_crossentropy",optimizer=opt,metrics=["accuracy"])
#model.fit_generator(train_batches, steps_per_epoch=16, validation_data=valid_batche
s, validation_steps=4, epochs=10, verbose=1)
```

In [15]:
```
opt = tf.keras.optimizers.Adam(lr=_learningRate)

mcp_save =  tf.keras.callbacks.ModelCheckpoint(_bestSaveModelName, save_best_only=True, monitor='val_loss', mode='min')

model.compile(optimizer=opt, loss=keras.losses.categorical_crossentropy, metrics=['accuracy'])
history = model.fit_generator(train_batches,  validation_data=valid_batches, epochs=_epochs, verbose=1, callbacks=[mcp_save])
```

```
Epoch 1/100
4/4 [==============================] - 22s 6s/step - loss: 1.1264 - accuracy: 0.2750
- val_loss: 1.0937 - val_accuracy: 0.3333
Epoch 2/100
4/4 [==============================] - 34s 8s/step - loss: 1.0876 - accuracy: 0.4333
- val_loss: 1.0758 - val_accuracy: 0.3333
Epoch 3/100
4/4 [==============================] - 38s 9s/step - loss: 1.0315 - accuracy: 0.4250
- val_loss: 1.0096 - val_accuracy: 0.5333
Epoch 4/100
4/4 [==============================] - 4s 932ms/step - loss: 0.9610 - accuracy: 0.55
00 - val_loss: 1.1263 - val_accuracy: 0.4000
Epoch 5/100
4/4 [==============================] - 35s 9s/step - loss: 0.9585 - accuracy: 0.4583
- val_loss: 0.8561 - val_accuracy: 0.6333
Epoch 6/100
4/4 [==============================] - 39s 10s/step - loss: 0.8255 - accuracy: 0.650
0 - val_loss: 0.7784 - val_accuracy: 0.6333
Epoch 7/100
4/4 [==============================] - 39s 10s/step - loss: 0.7629 - accuracy: 0.625
0 - val_loss: 0.6698 - val_accuracy: 0.7000
Epoch 8/100
4/4 [==============================] - 40s 10s/step - loss: 0.6779 - accuracy: 0.616
7 - val_loss: 0.6643 - val_accuracy: 0.6667
Epoch 9/100
4/4 [==============================] - 39s 10s/step - loss: 0.6159 - accuracy: 0.650
0 - val_loss: 0.5726 - val_accuracy: 0.6000
Epoch 10/100
4/4 [==============================] - 39s 10s/step - loss: 0.5671 - accuracy: 0.650
0 - val_loss: 0.5485 - val_accuracy: 0.6667
Epoch 11/100
4/4 [==============================] - 4s 881ms/step - loss: 0.5526 - accuracy: 0.70
00 - val_loss: 0.5541 - val_accuracy: 0.7000
Epoch 12/100
4/4 [==============================] - 33s 8s/step - loss: 0.5085 - accuracy: 0.7167
- val_loss: 0.4685 - val_accuracy: 0.8333
Epoch 13/100
4/4 [==============================] - 40s 10s/step - loss: 0.4976 - accuracy: 0.708
3 - val_loss: 0.4219 - val_accuracy: 0.7333
Epoch 14/100
4/4 [==============================] - 3s 834ms/step - loss: 0.4730 - accuracy: 0.75
00 - val_loss: 0.5732 - val_accuracy: 0.7333
Epoch 15/100
4/4 [==============================] - 34s 9s/step - loss: 0.6346 - accuracy: 0.6750
- val_loss: 0.3689 - val_accuracy: 0.9000
Epoch 16/100
4/4 [==============================] - 4s 992ms/step - loss: 0.5345 - accuracy: 0.67
50 - val_loss: 0.3980 - val_accuracy: 0.7667
Epoch 17/100
4/4 [==============================] - 3s 855ms/step - loss: 0.4664 - accuracy: 0.78
33 - val_loss: 0.4186 - val_accuracy: 0.6667
Epoch 18/100
4/4 [==============================] - 14s 4s/step - loss: 0.4635 - accuracy: 0.7417
- val_loss: 0.3916 - val_accuracy: 0.7667
Epoch 19/100
4/4 [==============================] - 48s 12s/step - loss: 0.4401 - accuracy: 0.741
7 - val_loss: 0.6837 - val_accuracy: 0.8667
Epoch 20/100
4/4 [==============================] - 31s 8s/step - loss: 0.3942 - accuracy: 0.8500
- val_loss: 0.4533 - val_accuracy: 0.9333
Epoch 21/100
4/4 [==============================] - 41s 10s/step - loss: 0.3525 - accuracy: 0.858
3 - val_loss: 0.3299 - val_accuracy: 0.9000
```

```
Epoch 22/100
4/4 [==============================] - 4s 1s/step - loss: 0.5262 - accuracy: 0.7500
- val_loss: 0.7232 - val_accuracy: 0.8667
Epoch 23/100
4/4 [==============================] - 4s 907ms/step - loss: 0.3608 - accuracy: 0.82
50 - val_loss: 0.4020 - val_accuracy: 0.8667
Epoch 24/100
4/4 [==============================] - 29s 7s/step - loss: 0.3199 - accuracy: 0.8917
- val_loss: 0.3236 - val_accuracy: 0.8000
Epoch 25/100
4/4 [==============================] - 35s 9s/step - loss: 0.3691 - accuracy: 0.8250
- val_loss: 0.2953 - val_accuracy: 0.9000
Epoch 26/100
4/4 [==============================] - 39s 10s/step - loss: 0.3766 - accuracy: 0.808
3 - val_loss: 0.2506 - val_accuracy: 0.9333
Epoch 27/100
4/4 [==============================] - 38s 10s/step - loss: 0.2771 - accuracy: 0.891
7 - val_loss: 0.2092 - val_accuracy: 0.9333
Epoch 28/100
4/4 [==============================] - 40s 10s/step - loss: 0.2582 - accuracy: 0.916
7 - val_loss: 0.1496 - val_accuracy: 0.9333
Epoch 29/100
4/4 [==============================] - 4s 938ms/step - loss: 0.2607 - accuracy: 0.87
50 - val_loss: 0.1817 - val_accuracy: 0.9667
Epoch 30/100
4/4 [==============================] - 4s 948ms/step - loss: 0.2616 - accuracy: 0.89
17 - val_loss: 0.2454 - val_accuracy: 0.9333
Epoch 31/100
4/4 [==============================] - 4s 877ms/step - loss: 0.2902 - accuracy: 0.88
33 - val_loss: 0.1730 - val_accuracy: 0.9333
Epoch 32/100
4/4 [==============================] - 4s 910ms/step - loss: 0.2318 - accuracy: 0.88
33 - val_loss: 0.2902 - val_accuracy: 0.9000
Epoch 33/100
4/4 [==============================] - 3s 801ms/step - loss: 0.2632 - accuracy: 0.91
67 - val_loss: 0.4056 - val_accuracy: 0.9000
Epoch 34/100
4/4 [==============================] - 4s 888ms/step - loss: 0.2279 - accuracy: 0.93
33 - val_loss: 0.4149 - val_accuracy: 0.9333
Epoch 35/100
4/4 [==============================] - 3s 775ms/step - loss: 0.2394 - accuracy: 0.89
17 - val_loss: 0.4100 - val_accuracy: 0.9333
Epoch 36/100
4/4 [==============================] - 4s 908ms/step - loss: 0.2181 - accuracy: 0.89
17 - val_loss: 0.3223 - val_accuracy: 0.9000
Epoch 37/100
4/4 [==============================] - 3s 817ms/step - loss: 0.1685 - accuracy: 0.90
83 - val_loss: 0.3053 - val_accuracy: 0.9333
Epoch 38/100
4/4 [==============================] - 3s 835ms/step - loss: 0.1913 - accuracy: 0.92
50 - val_loss: 0.1841 - val_accuracy: 0.9333
Epoch 39/100
4/4 [==============================] - 3s 762ms/step - loss: 0.2235 - accuracy: 0.88
33 - val_loss: 0.1548 - val_accuracy: 0.9667
Epoch 40/100
4/4 [==============================] - 3s 717ms/step - loss: 0.1815 - accuracy: 0.93
33 - val_loss: 0.2712 - val_accuracy: 0.9333
Epoch 41/100
4/4 [==============================] - 4s 924ms/step - loss: 0.1739 - accuracy: 0.91
67 - val_loss: 0.3831 - val_accuracy: 0.8667
Epoch 42/100
4/4 [==============================] - 3s 819ms/step - loss: 0.2032 - accuracy: 0.89
17 - val_loss: 0.3390 - val_accuracy: 0.9000
```

```
Epoch 43/100
4/4 [==============================] - 3s 799ms/step - loss: 0.2178 - accuracy: 0.91
67 - val_loss: 0.2200 - val_accuracy: 0.9667
Epoch 44/100
4/4 [==============================] - 3s 836ms/step - loss: 0.2428 - accuracy: 0.86
67 - val_loss: 0.5805 - val_accuracy: 0.9000
Epoch 45/100
4/4 [==============================] - 3s 866ms/step - loss: 0.1921 - accuracy: 0.90
83 - val_loss: 0.4188 - val_accuracy: 0.9000
Epoch 46/100
4/4 [==============================] - 4s 891ms/step - loss: 0.1866 - accuracy: 0.90
00 - val_loss: 0.5072 - val_accuracy: 0.9000
Epoch 47/100
4/4 [==============================] - 3s 854ms/step - loss: 0.2100 - accuracy: 0.90
83 - val_loss: 0.1535 - val_accuracy: 0.9333
Epoch 48/100
4/4 [==============================] - 21s 5s/step - loss: 0.1470 - accuracy: 0.9333
- val_loss: 0.1213 - val_accuracy: 0.9333
Epoch 49/100
4/4 [==============================] - 34s 8s/step - loss: 0.1236 - accuracy: 0.9333
- val_loss: 0.1194 - val_accuracy: 0.9333
Epoch 50/100
4/4 [==============================] - 41s 10s/step - loss: 0.1481 - accuracy: 0.958
3 - val_loss: 0.0627 - val_accuracy: 1.0000
Epoch 51/100
4/4 [==============================] - 38s 10s/step - loss: 0.1672 - accuracy: 0.925
0 - val_loss: 0.0510 - val_accuracy: 1.0000
Epoch 52/100
4/4 [==============================] - 4s 982ms/step - loss: 0.1947 - accuracy: 0.93
33 - val_loss: 0.0967 - val_accuracy: 0.9667
Epoch 53/100
4/4 [==============================] - 4s 1s/step - loss: 0.1681 - accuracy: 0.9167
- val_loss: 0.1339 - val_accuracy: 0.9333
Epoch 54/100
4/4 [==============================] - 4s 996ms/step - loss: 0.2885 - accuracy: 0.86
67 - val_loss: 0.5296 - val_accuracy: 0.8667
Epoch 55/100
4/4 [==============================] - 4s 947ms/step - loss: 0.1958 - accuracy: 0.93
33 - val_loss: 1.0038 - val_accuracy: 0.8667
Epoch 56/100
4/4 [==============================] - 4s 966ms/step - loss: 0.2913 - accuracy: 0.85
00 - val_loss: 1.0915 - val_accuracy: 0.9333
Epoch 57/100
4/4 [==============================] - 3s 831ms/step - loss: 0.1830 - accuracy: 0.91
67 - val_loss: 1.1960 - val_accuracy: 0.9000
Epoch 58/100
4/4 [==============================] - 4s 888ms/step - loss: 0.1818 - accuracy: 0.91
67 - val_loss: 0.1871 - val_accuracy: 0.9333
Epoch 59/100
4/4 [==============================] - 4s 952ms/step - loss: 0.1771 - accuracy: 0.93
33 - val_loss: 0.3573 - val_accuracy: 0.9333
Epoch 60/100
4/4 [==============================] - 4s 888ms/step - loss: 0.1630 - accuracy: 0.93
33 - val_loss: 0.3929 - val_accuracy: 0.9000
Epoch 61/100
4/4 [==============================] - 3s 847ms/step - loss: 0.1264 - accuracy: 0.94
17 - val_loss: 0.4938 - val_accuracy: 0.9000
Epoch 62/100
4/4 [==============================] - 3s 872ms/step - loss: 0.1140 - accuracy: 0.95
83 - val_loss: 0.6114 - val_accuracy: 0.9000
Epoch 63/100
4/4 [==============================] - 3s 783ms/step - loss: 0.1154 - accuracy: 0.93
33 - val_loss: 1.2162 - val_accuracy: 0.9000
```

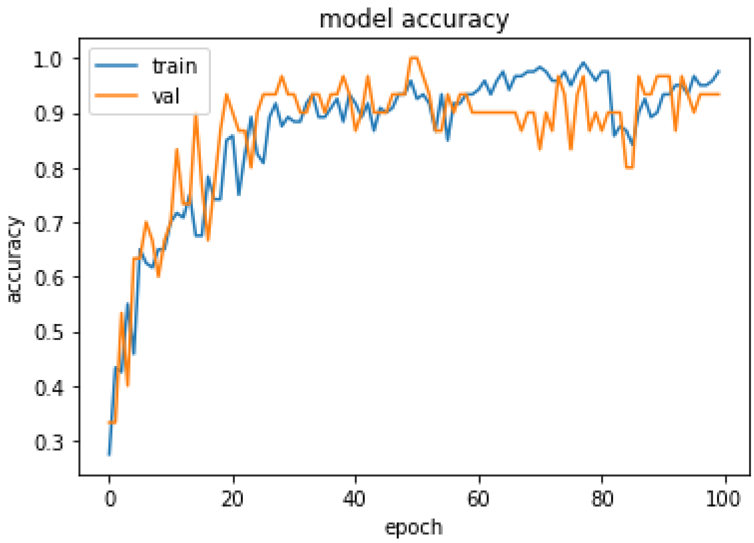
```
Epoch 64/100
4/4 [==============================] - 3s 844ms/step - loss: 0.1063 - accuracy: 0.95
83 - val_loss: 1.0034 - val_accuracy: 0.9000
Epoch 65/100
4/4 [==============================] - 3s 839ms/step - loss: 0.0820 - accuracy: 0.97
50 - val_loss: 1.0575 - val_accuracy: 0.9000
Epoch 66/100
4/4 [==============================] - 3s 843ms/step - loss: 0.1188 - accuracy: 0.94
17 - val_loss: 2.2247 - val_accuracy: 0.9000
Epoch 67/100
4/4 [==============================] - 4s 1s/step - loss: 0.0859 - accuracy: 0.9667
- val_loss: 1.6789 - val_accuracy: 0.9000
Epoch 68/100
4/4 [==============================] - 4s 888ms/step - loss: 0.0674 - accuracy: 0.96
67 - val_loss: 1.8637 - val_accuracy: 0.8667
Epoch 69/100
4/4 [==============================] - 3s 858ms/step - loss: 0.0570 - accuracy: 0.97
50 - val_loss: 1.6713 - val_accuracy: 0.9000
Epoch 70/100
4/4 [==============================] - 3s 845ms/step - loss: 0.0821 - accuracy: 0.97
50 - val_loss: 1.5420 - val_accuracy: 0.9000
Epoch 71/100
4/4 [==============================] - 3s 855ms/step - loss: 0.0495 - accuracy: 0.98
33 - val_loss: 1.4587 - val_accuracy: 0.8333
Epoch 72/100
4/4 [==============================] - 3s 809ms/step - loss: 0.0662 - accuracy: 0.97
50 - val_loss: 1.5145 - val_accuracy: 0.9000
Epoch 73/100
4/4 [==============================] - 3s 807ms/step - loss: 0.1061 - accuracy: 0.95
83 - val_loss: 1.7428 - val_accuracy: 0.8667
Epoch 74/100
4/4 [==============================] - 4s 907ms/step - loss: 0.0641 - accuracy: 0.95
83 - val_loss: 0.3655 - val_accuracy: 0.9667
Epoch 75/100
4/4 [==============================] - 3s 822ms/step - loss: 0.0678 - accuracy: 0.97
50 - val_loss: 0.5452 - val_accuracy: 0.9333
Epoch 76/100
4/4 [==============================] - 3s 812ms/step - loss: 0.0960 - accuracy: 0.95
00 - val_loss: 1.6132 - val_accuracy: 0.8333
Epoch 77/100
4/4 [==============================] - 4s 893ms/step - loss: 0.0795 - accuracy: 0.97
50 - val_loss: 0.6466 - val_accuracy: 0.9333
Epoch 78/100
4/4 [==============================] - 3s 753ms/step - loss: 0.0470 - accuracy: 0.99
17 - val_loss: 0.2600 - val_accuracy: 0.9667
Epoch 79/100
4/4 [==============================] - 3s 828ms/step - loss: 0.0551 - accuracy: 0.97
50 - val_loss: 0.6218 - val_accuracy: 0.8667
Epoch 80/100
4/4 [==============================] - 3s 754ms/step - loss: 0.0963 - accuracy: 0.95
83 - val_loss: 1.5088 - val_accuracy: 0.9000
Epoch 81/100
4/4 [==============================] - 4s 927ms/step - loss: 0.0604 - accuracy: 0.97
50 - val_loss: 2.0167 - val_accuracy: 0.8667
Epoch 82/100
4/4 [==============================] - 4s 933ms/step - loss: 0.0788 - accuracy: 0.97
50 - val_loss: 0.9017 - val_accuracy: 0.9000
Epoch 83/100
4/4 [==============================] - 3s 849ms/step - loss: 0.4276 - accuracy: 0.85
83 - val_loss: 1.4734 - val_accuracy: 0.9000
Epoch 84/100
4/4 [==============================] - 3s 826ms/step - loss: 0.2948 - accuracy: 0.87
50 - val_loss: 0.4746 - val_accuracy: 0.9000
```

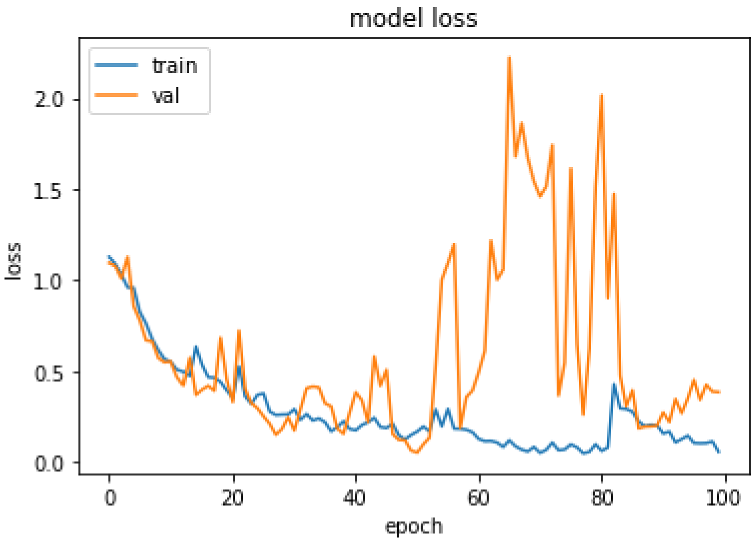
/

```
Epoch 85/100
4/4 [==============================] - 4s 913ms/step - loss: 0.2910 - accuracy: 0.86
67 - val_loss: 0.3071 - val_accuracy: 0.8000
Epoch 86/100
4/4 [==============================] - 3s 804ms/step - loss: 0.2763 - accuracy: 0.84
17 - val_loss: 0.3945 - val_accuracy: 0.8000
Epoch 87/100
4/4 [==============================] - 3s 870ms/step - loss: 0.2224 - accuracy: 0.90
00 - val_loss: 0.1843 - val_accuracy: 0.9667
Epoch 88/100
4/4 [==============================] - 3s 745ms/step - loss: 0.1971 - accuracy: 0.92
50 - val_loss: 0.1943 - val_accuracy: 0.9333
Epoch 89/100
4/4 [==============================] - 3s 768ms/step - loss: 0.2042 - accuracy: 0.89
17 - val_loss: 0.1958 - val_accuracy: 0.9333
Epoch 90/100
4/4 [==============================] - 4s 947ms/step - loss: 0.2016 - accuracy: 0.90
00 - val_loss: 0.1998 - val_accuracy: 0.9667
Epoch 91/100
4/4 [==============================] - 3s 817ms/step - loss: 0.1564 - accuracy: 0.93
33 - val_loss: 0.2721 - val_accuracy: 0.9667
Epoch 92/100
4/4 [==============================] - 3s 804ms/step - loss: 0.1663 - accuracy: 0.93
33 - val_loss: 0.2196 - val_accuracy: 0.9667
Epoch 93/100
4/4 [==============================] - 3s 869ms/step - loss: 0.1073 - accuracy: 0.95
00 - val_loss: 0.3457 - val_accuracy: 0.8667
Epoch 94/100
4/4 [==============================] - 3s 797ms/step - loss: 0.1252 - accuracy: 0.95
00 - val_loss: 0.2684 - val_accuracy: 0.9667
Epoch 95/100
4/4 [==============================] - 3s 862ms/step - loss: 0.1440 - accuracy: 0.93
33 - val_loss: 0.3491 - val_accuracy: 0.9333
Epoch 96/100
4/4 [==============================] - 4s 1s/step - loss: 0.1040 - accuracy: 0.9667
- val_loss: 0.4512 - val_accuracy: 0.9000
Epoch 97/100
4/4 [==============================] - 4s 897ms/step - loss: 0.1021 - accuracy: 0.95
00 - val_loss: 0.3420 - val_accuracy: 0.9333
Epoch 98/100
4/4 [==============================] - 4s 936ms/step - loss: 0.1033 - accuracy: 0.95
00 - val_loss: 0.4247 - val_accuracy: 0.9333
Epoch 99/100
4/4 [==============================] - 3s 822ms/step - loss: 0.1121 - accuracy: 0.95
83 - val_loss: 0.3874 - val_accuracy: 0.9333
Epoch 100/100
4/4 [==============================] - 3s 795ms/step - loss: 0.0559 - accuracy: 0.97
50 - val_loss: 0.3844 - val_accuracy: 0.9333
```

A very erratic validation loss can be seen but the accuracy climbs up steadily and plateaus around 95%

In [16]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```





Check Accuracy against 30 test images

/

In [17]:
```
test_batches = ImageDataGenerator(rescale=1. / 255,
    shear_range=0.5,
    zoom_range=0.5,
    rotation_range=45,
    width_shift_range=.15,
    height_shift_range=.15,
    horizontal_flip=True,
    vertical_flip=True).flow_from_directory(test_data_dir, target_size=(224,224), bat
ch_size=10)
d = model.evaluate_generator(test_batches)
print("Loss: {} and Accuracy: {:02}".format(d[0],d[1]*100))
```

```
Found 29 images belonging to 3 classes.
Loss: 0.4342779219150543 and Accuracy: 86.20689511299133
```

In [30]:
```
tr = model.evaluate_generator(train_batches)
v = model.evaluate_generator(valid_batches)
test = model.evaluate_generator(test_batches)

print("Train Loss: {} and Accuracy: {:02}".format(tr[0],tr[1]*100))
print("Valid Loss: {} and Accuracy: {:02}".format(v[0],v[1]*100))
print("Test Loss: {} and Accuracy: {:02}".format(test[0],test[1]*100))
```

```
Train Loss: 0.06878527253866196 and Accuracy: 96.66666388511658
Valid Loss: 0.4679800570011139 and Accuracy: 93.33333373069763
Test Loss: 0.5713502168655396 and Accuracy: 75.86206793785095
```

Poor test accuracy. Over fitting could be a problem. Too complex layers might result in overfitting

In [ ]:
```
model.save("VGGbananaM1.h5")
```

Checking for random image to test

In [63]:
```
from keras.preprocessing import image
from IPython.display import Image


import numpy as np
img = image.load_img('../input/bananacathand/Dataset/train/cat/cat.23.jpg', target_si
ze=(224,224))
display(img)
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = x*(1./255)
images = np.vstack([x])

classes = model.predict(images)
print(classes*100)
print(valid_batches.class_indices)
print(_classes[np.argmax(classes)])
```

```
[[2.2680389e-03 9.9996552e+01 1.1849966e-03]]
{'banana': 0, 'cat': 1, 'hand': 2}
cat
```

In [ ]:
```
#For saving the model


# print(tf.__version__)
# from keras.models import model_from_json
# loaded_model = tf.keras.models.load_model("best.h5")
# json_model = loaded_model.to_json()
# with open('BananaWeights_model.json', 'w') as json_file:
#     json_file.write(json_model)
# #saving the weights of the model
# loaded_model.save_weights('BananaWeights.h5')
```

In [ ]:
```
# json_file = open('BananaWeights_model.json', 'r')
# loaded_model_json = json_file.read()
# json_file.close()
# loaded_model = model_from_json(loaded_model_json)
# # load weights into new model
# loaded_model.load_weights("BananaWeights.h5")
# loaded_model.compile(optimizer=opt, loss=keras.losses.categorical_crossentropy, met
rics=['accuracy'])
# loaded_model.evaluate_generator(train_batches)
```

Now I test with the test set that is out of the 150 training images to get an idea

In [36]:
```
print(model)
from keras.preprocessing import image
import numpy as np

for c in os.listdir("../input/bananacathand/Dataset/test/cat"):
  print(c)
  img = image.load_img("../input/bananacathand/Dataset/test/cat/" +c, target_size=(22
4,224))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)
  x = x*(1./255)
  images = np.vstack([x])
  classes = model.predict(images)
  print(_classes[np.argmax(classes)])
  print()
```

```
<tensorflow.python.keras.engine.sequential.Sequential object at 0x7f6bd1662dd0>
cat.25.jpg
cat

cat.8.jpg
cat

cat.36.jpg
cat

rscat2.jpg
cat

vatx.jpg
cat

cat.17.jpg
cat

cat-banana.jpg
banana

cat.26.jpg
hand

cat.44.jpg
cat

cat.13.jpg
cat

cat.15.jpg
cat

cat.11.jpg
cat

cat.3.jpg
hand

cathand.jpg
hand
```

Trying to reduce complexity to achieve faster training speed and lesser model size

I feel this model is too complex for the need. Thus need to reduce layers to improve the problem of overfitting. Trying to play around with filters

In [38]:

```
model2 = tf.keras.Sequential()

model2.add(Conv2D(input_shape=(224,224,3),filters=32,kernel_size=(3,3),padding="same"
, activation="relu"))
model2.add(Conv2D(filters=32,kernel_size=(3,3),padding="same", activation="relu"))
model2.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))


model2.add(Conv2D(filters=64, kernel_size=(3,3), padding="same", activation="relu"))
model2.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model2.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model2.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model2.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model2.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model2.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model2.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model2.add(Flatten())
model2.add(Dense(units=1024,activation="relu"))
model2.add(Dense(units=1024,activation="relu"))

model2.add(Dense(units=3, activation="softmax"))
```

In [39]:

```
_bestSaveModelName2 = "best2.h5"
opt = tf.keras.optimizers.Adam(lr=_learningRate)

mcp_save =  tf.keras.callbacks.ModelCheckpoint(_bestSaveModelName2, save_best_only=True, monitor='val_loss', mode='min')

model2.compile(optimizer=opt, loss=keras.losses.categorical_crossentropy, metrics=['accuracy'])
history = model2.fit_generator(train_batches,  validation_data=valid_batches, epochs=_epochs, verbose=1, callbacks=[mcp_save])
```

```
Epoch 1/100
4/4 [==============================] - 5s 1s/step - loss: 1.0986 - accuracy: 0.3250
- val_loss: 1.0883 - val_accuracy: 0.3333
Epoch 2/100
4/4 [==============================] - 5s 1s/step - loss: 1.0818 - accuracy: 0.3333
- val_loss: 1.0691 - val_accuracy: 0.6333
Epoch 3/100
4/4 [==============================] - 5s 1s/step - loss: 1.0674 - accuracy: 0.4917
- val_loss: 1.0414 - val_accuracy: 0.6667
Epoch 4/100
4/4 [==============================] - 5s 1s/step - loss: 0.9774 - accuracy: 0.5500
- val_loss: 0.9840 - val_accuracy: 0.6000
Epoch 5/100
4/4 [==============================] - 6s 2s/step - loss: 0.9079 - accuracy: 0.6083
- val_loss: 0.9058 - val_accuracy: 0.6333
Epoch 6/100
4/4 [==============================] - 8s 2s/step - loss: 0.8675 - accuracy: 0.5333
- val_loss: 0.7517 - val_accuracy: 0.7000
Epoch 7/100
4/4 [==============================] - 13s 3s/step - loss: 0.7699 - accuracy: 0.6083
- val_loss: 0.6746 - val_accuracy: 0.6667
Epoch 8/100
4/4 [==============================] - 5s 1s/step - loss: 0.7202 - accuracy: 0.6250
- val_loss: 0.6703 - val_accuracy: 0.6333
Epoch 9/100
4/4 [==============================] - 13s 3s/step - loss: 0.6604 - accuracy: 0.7000
- val_loss: 0.5811 - val_accuracy: 0.7667
Epoch 10/100
4/4 [==============================] - 6s 1s/step - loss: 0.6051 - accuracy: 0.7333
- val_loss: 0.5572 - val_accuracy: 0.6333
Epoch 11/100
4/4 [==============================] - 13s 3s/step - loss: 0.5683 - accuracy: 0.6833
- val_loss: 0.5118 - val_accuracy: 0.9333
Epoch 12/100
4/4 [==============================] - 5s 1s/step - loss: 0.5232 - accuracy: 0.7500
- val_loss: 0.4465 - val_accuracy: 0.6667
Epoch 13/100
4/4 [==============================] - 3s 821ms/step - loss: 0.5417 - accuracy: 0.68
33 - val_loss: 0.4525 - val_accuracy: 0.8000
Epoch 14/100
4/4 [==============================] - 5s 1s/step - loss: 0.4989 - accuracy: 0.7917
- val_loss: 0.3535 - val_accuracy: 0.9333
Epoch 15/100
4/4 [==============================] - 4s 876ms/step - loss: 0.5088 - accuracy: 0.71
67 - val_loss: 0.3629 - val_accuracy: 0.9333
Epoch 16/100
4/4 [==============================] - 5s 1s/step - loss: 0.4493 - accuracy: 0.8000
- val_loss: 0.3481 - val_accuracy: 0.9667
Epoch 17/100
4/4 [==============================] - 13s 3s/step - loss: 0.4152 - accuracy: 0.7917
- val_loss: 0.2591 - val_accuracy: 0.9667
Epoch 18/100
4/4 [==============================] - 3s 827ms/step - loss: 0.3623 - accuracy: 0.85
83 - val_loss: 0.2900 - val_accuracy: 0.8667
Epoch 19/100
4/4 [==============================] - 3s 855ms/step - loss: 0.3871 - accuracy: 0.80
83 - val_loss: 0.2979 - val_accuracy: 0.9333
Epoch 20/100
4/4 [==============================] - 5s 1s/step - loss: 0.3586 - accuracy: 0.8250
- val_loss: 0.1820 - val_accuracy: 1.0000
Epoch 21/100
4/4 [==============================] - 4s 887ms/step - loss: 0.3271 - accuracy: 0.85
83 - val_loss: 0.2786 - val_accuracy: 0.9000
```

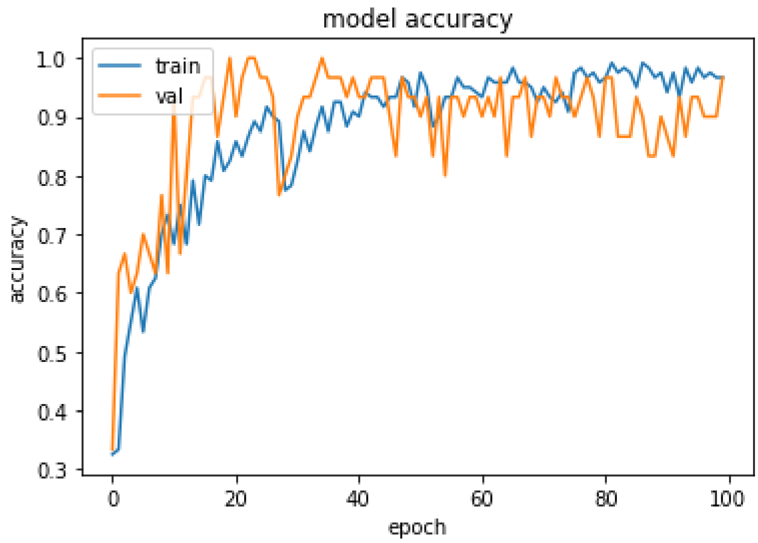
```
Epoch 22/100
4/4 [==============================] - 5s 1s/step - loss: 0.3780 - accuracy: 0.8333
- val_loss: 0.1777 - val_accuracy: 0.9667
Epoch 23/100
4/4 [==============================] - 5s 1s/step - loss: 0.2803 - accuracy: 0.8667
- val_loss: 0.1148 - val_accuracy: 1.0000
Epoch 24/100
4/4 [==============================] - 5s 1s/step - loss: 0.2566 - accuracy: 0.8917
- val_loss: 0.0793 - val_accuracy: 1.0000
Epoch 25/100
4/4 [==============================] - 4s 925ms/step - loss: 0.2561 - accuracy: 0.87
50 - val_loss: 0.1270 - val_accuracy: 0.9667
Epoch 26/100
4/4 [==============================] - 4s 1s/step - loss: 0.2040 - accuracy: 0.9167
- val_loss: 0.0944 - val_accuracy: 0.9667
Epoch 27/100
4/4 [==============================] - 3s 848ms/step - loss: 0.2404 - accuracy: 0.90
00 - val_loss: 0.1051 - val_accuracy: 0.9333
Epoch 28/100
4/4 [==============================] - 4s 878ms/step - loss: 0.3156 - accuracy: 0.89
17 - val_loss: 0.5875 - val_accuracy: 0.7667
Epoch 29/100
4/4 [==============================] - 4s 926ms/step - loss: 0.5289 - accuracy: 0.77
50 - val_loss: 0.4203 - val_accuracy: 0.8000
Epoch 30/100
4/4 [==============================] - 3s 859ms/step - loss: 0.5357 - accuracy: 0.78
33 - val_loss: 0.3876 - val_accuracy: 0.8333
Epoch 31/100
4/4 [==============================] - 4s 936ms/step - loss: 0.3941 - accuracy: 0.82
50 - val_loss: 0.3581 - val_accuracy: 0.9000
Epoch 32/100
4/4 [==============================] - 3s 850ms/step - loss: 0.3744 - accuracy: 0.87
50 - val_loss: 0.2765 - val_accuracy: 0.9333
Epoch 33/100
4/4 [==============================] - 4s 971ms/step - loss: 0.3333 - accuracy: 0.84
17 - val_loss: 0.2494 - val_accuracy: 0.9333
Epoch 34/100
4/4 [==============================] - 3s 874ms/step - loss: 0.2699 - accuracy: 0.88
33 - val_loss: 0.1337 - val_accuracy: 0.9667
Epoch 35/100
4/4 [==============================] - 3s 746ms/step - loss: 0.2215 - accuracy: 0.91
67 - val_loss: 0.1143 - val_accuracy: 1.0000
Epoch 36/100
4/4 [==============================] - 5s 1s/step - loss: 0.2528 - accuracy: 0.8750
- val_loss: 0.0658 - val_accuracy: 0.9667
Epoch 37/100
4/4 [==============================] - 4s 893ms/step - loss: 0.1857 - accuracy: 0.92
50 - val_loss: 0.1448 - val_accuracy: 0.9667
Epoch 38/100
4/4 [==============================] - 4s 920ms/step - loss: 0.1733 - accuracy: 0.92
50 - val_loss: 0.1012 - val_accuracy: 0.9667
Epoch 39/100
4/4 [==============================] - 4s 884ms/step - loss: 0.2171 - accuracy: 0.88
33 - val_loss: 0.1676 - val_accuracy: 0.9333
Epoch 40/100
4/4 [==============================] - 5s 1s/step - loss: 0.2099 - accuracy: 0.9083
- val_loss: 0.1373 - val_accuracy: 0.9667
Epoch 41/100
4/4 [==============================] - 3s 870ms/step - loss: 0.1776 - accuracy: 0.90
00 - val_loss: 0.1662 - val_accuracy: 0.9333
Epoch 42/100
4/4 [==============================] - 3s 782ms/step - loss: 0.1561 - accuracy: 0.94
17 - val_loss: 0.2147 - val_accuracy: 0.9333
```

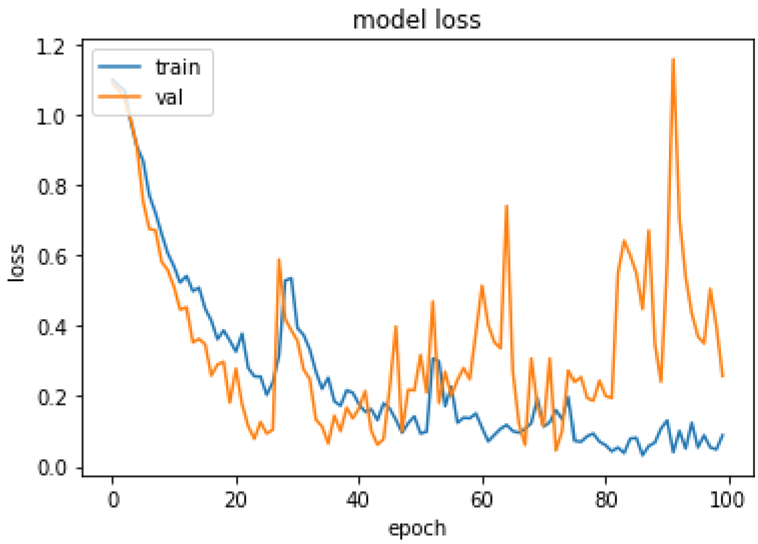
```
Epoch 43/100
4/4 [==============================] - 4s 909ms/step - loss: 0.1630 - accuracy: 0.93
33 - val_loss: 0.1023 - val_accuracy: 0.9667
Epoch 44/100
4/4 [==============================] - 6s 1s/step - loss: 0.1324 - accuracy: 0.9333
- val_loss: 0.0623 - val_accuracy: 0.9667
Epoch 45/100
4/4 [==============================] - 4s 941ms/step - loss: 0.1801 - accuracy: 0.91
67 - val_loss: 0.0784 - val_accuracy: 0.9667
Epoch 46/100
4/4 [==============================] - 3s 768ms/step - loss: 0.1652 - accuracy: 0.93
33 - val_loss: 0.2259 - val_accuracy: 0.9000
Epoch 47/100
4/4 [==============================] - 3s 725ms/step - loss: 0.1332 - accuracy: 0.93
33 - val_loss: 0.3982 - val_accuracy: 0.8333
Epoch 48/100
4/4 [==============================] - 4s 929ms/step - loss: 0.0964 - accuracy: 0.96
67 - val_loss: 0.1003 - val_accuracy: 0.9667
Epoch 49/100
4/4 [==============================] - 3s 830ms/step - loss: 0.1236 - accuracy: 0.95
83 - val_loss: 0.2188 - val_accuracy: 0.9333
Epoch 50/100
4/4 [==============================] - 4s 885ms/step - loss: 0.1432 - accuracy: 0.91
67 - val_loss: 0.2168 - val_accuracy: 0.9333
Epoch 51/100
4/4 [==============================] - 3s 777ms/step - loss: 0.0944 - accuracy: 0.97
50 - val_loss: 0.3174 - val_accuracy: 0.9000
Epoch 52/100
4/4 [==============================] - 3s 834ms/step - loss: 0.0998 - accuracy: 0.95
00 - val_loss: 0.2104 - val_accuracy: 0.9333
Epoch 53/100
4/4 [==============================] - 4s 890ms/step - loss: 0.3075 - accuracy: 0.88
33 - val_loss: 0.4697 - val_accuracy: 0.8333
Epoch 54/100
4/4 [==============================] - 3s 838ms/step - loss: 0.2995 - accuracy: 0.90
00 - val_loss: 0.1811 - val_accuracy: 0.9333
Epoch 55/100
4/4 [==============================] - 3s 759ms/step - loss: 0.1722 - accuracy: 0.93
33 - val_loss: 0.2699 - val_accuracy: 0.8000
Epoch 56/100
4/4 [==============================] - 3s 838ms/step - loss: 0.2272 - accuracy: 0.93
33 - val_loss: 0.2017 - val_accuracy: 0.9333
Epoch 57/100
4/4 [==============================] - 4s 875ms/step - loss: 0.1247 - accuracy: 0.96
67 - val_loss: 0.2463 - val_accuracy: 0.9333
Epoch 58/100
4/4 [==============================] - 4s 985ms/step - loss: 0.1392 - accuracy: 0.95
00 - val_loss: 0.2804 - val_accuracy: 0.9000
Epoch 59/100
4/4 [==============================] - 4s 959ms/step - loss: 0.1376 - accuracy: 0.95
00 - val_loss: 0.2485 - val_accuracy: 0.9333
Epoch 60/100
4/4 [==============================] - 4s 922ms/step - loss: 0.1512 - accuracy: 0.94
17 - val_loss: 0.3811 - val_accuracy: 0.9333
Epoch 61/100
4/4 [==============================] - 3s 800ms/step - loss: 0.1106 - accuracy: 0.93
33 - val_loss: 0.5145 - val_accuracy: 0.9000
Epoch 62/100
4/4 [==============================] - 3s 841ms/step - loss: 0.0718 - accuracy: 0.96
67 - val_loss: 0.4034 - val_accuracy: 0.9333
Epoch 63/100
4/4 [==============================] - 4s 894ms/step - loss: 0.0902 - accuracy: 0.95
83 - val_loss: 0.3541 - val_accuracy: 0.9000
```

```
Epoch 64/100
4/4 [==============================] - 3s 766ms/step - loss: 0.1076 - accuracy: 0.95
83 - val_loss: 0.3365 - val_accuracy: 0.9667
Epoch 65/100
4/4 [==============================] - 3s 867ms/step - loss: 0.1186 - accuracy: 0.95
83 - val_loss: 0.7398 - val_accuracy: 0.8333
Epoch 66/100
4/4 [==============================] - 4s 947ms/step - loss: 0.1011 - accuracy: 0.98
33 - val_loss: 0.2717 - val_accuracy: 0.9333
Epoch 67/100
4/4 [==============================] - 4s 934ms/step - loss: 0.0965 - accuracy: 0.95
83 - val_loss: 0.1224 - val_accuracy: 0.9333
Epoch 68/100
4/4 [==============================] - 5s 1s/step - loss: 0.1071 - accuracy: 0.9583
- val_loss: 0.0615 - val_accuracy: 0.9667
Epoch 69/100
4/4 [==============================] - 4s 923ms/step - loss: 0.1239 - accuracy: 0.95
00 - val_loss: 0.3079 - val_accuracy: 0.8667
Epoch 70/100
4/4 [==============================] - 3s 848ms/step - loss: 0.1945 - accuracy: 0.92
50 - val_loss: 0.1737 - val_accuracy: 0.9333
Epoch 71/100
4/4 [==============================] - 3s 872ms/step - loss: 0.1135 - accuracy: 0.95
00 - val_loss: 0.1155 - val_accuracy: 0.9333
Epoch 72/100
4/4 [==============================] - 4s 901ms/step - loss: 0.1262 - accuracy: 0.93
33 - val_loss: 0.3075 - val_accuracy: 0.9000
Epoch 73/100
4/4 [==============================] - 6s 1s/step - loss: 0.1612 - accuracy: 0.9250
- val_loss: 0.0459 - val_accuracy: 0.9667
Epoch 74/100
4/4 [==============================] - 4s 944ms/step - loss: 0.1351 - accuracy: 0.94
17 - val_loss: 0.1007 - val_accuracy: 0.9333
Epoch 75/100
4/4 [==============================] - 4s 928ms/step - loss: 0.1979 - accuracy: 0.90
83 - val_loss: 0.2734 - val_accuracy: 0.9333
Epoch 76/100
4/4 [==============================] - 3s 827ms/step - loss: 0.0733 - accuracy: 0.97
50 - val_loss: 0.2411 - val_accuracy: 0.9000
Epoch 77/100
4/4 [==============================] - 3s 754ms/step - loss: 0.0707 - accuracy: 0.98
33 - val_loss: 0.2545 - val_accuracy: 0.9333
Epoch 78/100
4/4 [==============================] - 4s 887ms/step - loss: 0.0864 - accuracy: 0.96
67 - val_loss: 0.1951 - val_accuracy: 0.9667
Epoch 79/100
4/4 [==============================] - 12s 3s/step - loss: 0.0942 - accuracy: 0.9750
- val_loss: 0.1874 - val_accuracy: 0.9333
Epoch 80/100
4/4 [==============================] - 4s 875ms/step - loss: 0.0711 - accuracy: 0.95
83 - val_loss: 0.2443 - val_accuracy: 0.8667
Epoch 81/100
4/4 [==============================] - 4s 926ms/step - loss: 0.0610 - accuracy: 0.96
67 - val_loss: 0.2011 - val_accuracy: 0.9667
Epoch 82/100
4/4 [==============================] - 3s 832ms/step - loss: 0.0437 - accuracy: 0.99
17 - val_loss: 0.1952 - val_accuracy: 0.9667
Epoch 83/100
4/4 [==============================] - 4s 1s/step - loss: 0.0545 - accuracy: 0.9750
- val_loss: 0.5485 - val_accuracy: 0.8667
Epoch 84/100
4/4 [==============================] - 4s 900ms/step - loss: 0.0387 - accuracy: 0.98
33 - val_loss: 0.6406 - val_accuracy: 0.8667
```

```
Epoch 85/100
4/4 [==============================] - 4s 913ms/step - loss: 0.0798 - accuracy: 0.97
50 - val_loss: 0.5991 - val_accuracy: 0.8667
Epoch 86/100
4/4 [==============================] - 4s 975ms/step - loss: 0.0818 - accuracy: 0.95
00 - val_loss: 0.5482 - val_accuracy: 0.9333
Epoch 87/100
4/4 [==============================] - 3s 805ms/step - loss: 0.0325 - accuracy: 0.99
17 - val_loss: 0.4480 - val_accuracy: 0.9000
Epoch 88/100
4/4 [==============================] - 4s 955ms/step - loss: 0.0586 - accuracy: 0.98
33 - val_loss: 0.6700 - val_accuracy: 0.8333
Epoch 89/100
4/4 [==============================] - 3s 874ms/step - loss: 0.0683 - accuracy: 0.96
67 - val_loss: 0.3493 - val_accuracy: 0.8333
Epoch 90/100
4/4 [==============================] - 4s 891ms/step - loss: 0.1084 - accuracy: 0.97
50 - val_loss: 0.2413 - val_accuracy: 0.9000
Epoch 91/100
4/4 [==============================] - 4s 894ms/step - loss: 0.1313 - accuracy: 0.94
17 - val_loss: 0.5786 - val_accuracy: 0.8667
Epoch 92/100
4/4 [==============================] - 3s 846ms/step - loss: 0.0412 - accuracy: 0.97
50 - val_loss: 1.1572 - val_accuracy: 0.8333
Epoch 93/100
4/4 [==============================] - 4s 958ms/step - loss: 0.1017 - accuracy: 0.93
33 - val_loss: 0.7028 - val_accuracy: 0.9333
Epoch 94/100
4/4 [==============================] - 3s 871ms/step - loss: 0.0516 - accuracy: 0.98
33 - val_loss: 0.5390 - val_accuracy: 0.8667
Epoch 95/100
4/4 [==============================] - 4s 887ms/step - loss: 0.1245 - accuracy: 0.95
83 - val_loss: 0.4355 - val_accuracy: 0.9333
Epoch 96/100
4/4 [==============================] - 4s 880ms/step - loss: 0.0548 - accuracy: 0.98
33 - val_loss: 0.3702 - val_accuracy: 0.9333
Epoch 97/100
4/4 [==============================] - 3s 772ms/step - loss: 0.0891 - accuracy: 0.96
67 - val_loss: 0.3504 - val_accuracy: 0.9000
Epoch 98/100
4/4 [==============================] - 4s 903ms/step - loss: 0.0548 - accuracy: 0.97
50 - val_loss: 0.5050 - val_accuracy: 0.9000
Epoch 99/100
4/4 [==============================] - 3s 856ms/step - loss: 0.0491 - accuracy: 0.96
67 - val_loss: 0.3989 - val_accuracy: 0.9000
Epoch 100/100
4/4 [==============================] - 4s 898ms/step - loss: 0.0890 - accuracy: 0.96
67 - val_loss: 0.2578 - val_accuracy: 0.9667
```

In [65]:
```python
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```





In [47]:
```
bestModel = tf.keras.models.load_model("best2.h5")
d = bestModel.evaluate_generator(train_batches)
print("Train: Loss: {} and Accuracy: {:02}".format(d[0],d[1]*100))

d = bestModel.evaluate_generator(valid_batches)
print("Valid: Loss: {} and Accuracy: {:02}".format(d[0],d[1]*100))

d = bestModel.evaluate_generator(test_batches)
print("Test: Loss: {} and Accuracy: {:02}".format(d[0],d[1]*100))
```

```
Train: Loss: 0.17853008210659027 and Accuracy: 93.33333373069763
Valid: Loss: 0.05634515359997749 and Accuracy: 96.66666388511658
Test: Loss: 0.713640570640564 and Accuracy: 86.20689511299133
```

The model is achieveing better accuracy with test data set.

References:

1.https://keras.io/api/layers/ (https://keras.io/api/layers/)
2.https://www.kaggle.com/moltean/fruits? (https://www.kaggle.com/moltean/fruits?)
3.https://www.kaggle.com/c/dogs-vs-cats (https://www.kaggle.com/c/dogs-vs-cats)
4.https://neurohive.io/en/popular-networks/vgg16/ (https://neurohive.io/en/popular-networks/vgg16/)
5.https://www.geeksforgeeks.org/vgg-16-cnn-model/ (https://www.geeksforgeeks.org/vgg-16-cnn-model/)