Chihuahua or Muffin

1. Read data and preprocessing

```
[ ] from glob import glob
    import cv2

    import numpy as np
    import matplotlib.pyplot as plt
    import keras
```

```
[ ] filenames_chihuahua = glob("/content/gdrive/My Drive/Colab Notebooks/Chihuahua VS Muffin/ChihuahuaMuffin/*.jpg")
    filenames_muffin = glob("/content/gdrive/My Drive/Colab Notebooks/Chihuahua VS Muffin/ChihuahuaMuffin/*.jpeg")

    chihuahua_images = [cv2.resize(cv2.imread(img), (170,170)) for img in filenames_chihuahua if img.split('/')[-1] != 'full.jpg']
    muffin_images = [cv2.resize(cv2.imread(img), (170,170)) for img in filenames_muffin]
```

2. Generate more data, we flip images to have more image samples

```
[ ] chihuahua_images_horizontal_flip = [cv2.flip(img,0) for img in chihuahua_images]
    chihuahua_images_vertical_flip = [cv2.flip(img,1) for img in chihuahua_images]
    #chihuahua_images_greyscale = [cv2.cvtColor(img, cv2.COLOR_RGB2GRAY) for img in chihuahua_images]

    muffin_images_horizontal_flip = [cv2.flip(img,0) for img in muffin_images]
    muffin_images_vertical_flip = [cv2.flip(img,1) for img in muffin_images]
    #muffin_images_greyscale = [cv2.cvtColor(img, cv2.COLOR_RGB2GRAY) for img in muffin_images]
```

```
[ ] x = np.array(chihuahua_image + chihuahua_images_horizontal_flip + chihuahua_images_vertical_flip  +
                 muffin_image +muffin_images_horizontal_flip + muffin_images_vertical_flip ).astype('float32')/255
    y = keras.utils.to_categorical([0]*24 + [1]*24, num_classes=2)
```

```
[ ] x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=42)
```

3. Apply VGG16 with imagenet dataset, pre-trained model. We fixed the first 15 layers and train the rest. Then, perform fine tuning to get result. Optimizer that we used is RMSprop.

Using VGG16 and imagenet pretrain model

```
[ ] from keras.applications import VGG16
    from keras.layers import  Dropout, Dense,GlobalAveragePooling2D, Dense, Conv2D

    pre_trained_model = VGG16(include_top=False, weights = 'imagenet')
```

```
[ ] for layer in pre_trained_model.layers[:15]:
        layer.trainable = False

    for layer in pre_trained_model.layers[15:]:
        layer.trainable = True
```

```
    last_layer = pre_trained_model.layers[-1]
    last_output = last_layer.output

    # Flatten the output layer to 1 dimension
    x = GlobalAveragePooling2D()(last_output)
    # Add a fully connected layer with  ReLU activation
    x = Dense(128, activation='relu')(x)
    x = Dropout(0.25)(x)
    x = Dense(64, activation='relu')(x)
    x = Dropout(0.25)(x)
    # Add a final softmax layer for classification
    x = Dense(2, activation='softmax')(x)

    model = keras.models.Model(pre_trained_model.input, x)
    model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.RMSprop(lr=0.0001, decay=1e-6),  metrics=['accuracy'])
    model.summary()
```

4. Train the model and evaluate the result

```
[ ] model.fit(x_train, y_train, batch_size=10, epochs=10, verbose=1)
```

```
Epoch 1/10
32/32 [==============================] - 13s 391ms/step - loss: 0.8633 - acc: 0.5313
Epoch 2/10
32/32 [==============================] - 11s 340ms/step - loss: 0.4374 - acc: 0.7813
Epoch 3/10
32/32 [==============================] - 11s 338ms/step - loss: 0.3587 - acc: 0.7812
Epoch 4/10
32/32 [==============================] - 11s 338ms/step - loss: 0.1079 - acc: 1.0000
Epoch 5/10
32/32 [==============================] - 11s 338ms/step - loss: 0.0338 - acc: 0.9687
Epoch 6/10
32/32 [==============================] - 11s 339ms/step - loss: 0.0085 - acc: 1.0000
Epoch 7/10
32/32 [==============================] - 11s 338ms/step - loss: 0.0050 - acc: 1.0000
Epoch 8/10
32/32 [==============================] - 11s 339ms/step - loss: 0.0246 - acc: 1.0000
Epoch 9/10
32/32 [==============================] - 11s 338ms/step - loss: 0.0030 - acc: 1.0000
Epoch 10/10
32/32 [==============================] - 11s 338ms/step - loss: 0.0027 - acc: 1.0000
<keras.callbacks.History at 0x7f200f115160>
```

```
[ ] Y_pred = model.predict(x_test)
    acc = sum([np.argmax(y_test[i])==np.argmax(Y_pred[i]) for i in range(16)])/16
    print("Accuracy:",acc)
```

```
Accuracy: 1.0
```

5. Accuracy is 100%