Trabalho Final

Student: Daniel da Silva Costa

E-mail: danieldasilvacosta@gmail.com

This notebook was built based on the codes from:

- What does a CNN see? https://www.kaggle.com/code/aakashnain/what-does-a-cnn-see/notebook
- tf.keras.layers.Conv2D

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D

In [1]:
```
data_folder = './data/'
```

Imports

In [2]:
```
import os
import cv2
import glob
import imgaug as aug
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import imgaug.augmenters as iaa
from os import listdir
from pathlib import Path
from keras.models import Sequential, Model
# from keras.optimizers import Adam, SGD, RMSprop
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
from keras.callbacks import ModelCheckpoint, EarlyStopping
# from keras.utils import to_categorical
```

```
from tensorflow.keras.utils import to_categorical
from keras import backend as K
from keras.applications.vgg16 import preprocess_input
import tensorflow as tf
tf.compat.v1.disable_eager_execution()

color = sns.color_palette()
%matplotlib inline
%config InlineBackend.figure_format="svg"

from tensorflow.keras import layers, models
```

In [3]:
```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
Num GPUs Available:  1
```

In [4]:
```
print(tf.__version__)
```

```
2.6.0
```

Seed

In [5]:
```
os.environ['PYTHONHASHSEED'] = '0'

seed=1234

np.random.seed(seed)

tf.random.set_seed(seed)

aug.seed(seed)
```

Organising the training and validation data

In [6]:
```
training_data = Path(data_folder + '/training/')
validation_data = Path(data_folder + '/validation/')
```

```
labels_path = Path(data_folder + '/monkey_labels.txt')
```

In [7]:
```
labels_info = []

lines = labels_path.read_text().strip().splitlines()[1:]
for line in lines:
    line = line.split(',')
    line = [x.strip(' \n\t\r') for x in line]
    line[3], line[4] = int(line[3]), int(line[4])
    line = tuple(line)
    labels_info.append(line)

labels_info = pd.DataFrame(labels_info, columns=['Label', 'Latin Name', 'Common Name',
                                                  'Train Images', 'Validation Images'], index=None)
labels_info.head(10)
```

Out[7]:

	Label	Latin Name	Common Name	Train Images	Validation Images
0	n0	alouatta_palliata	mantled_howler	131	26
1	n1	erythrocebus_patas	patas_monkey	139	28
2	n2	cacajao_calvus	bald_uakari	137	27
3	n3	macaca_fuscata	japanese_macaque	152	30
4	n4	cebuella_pygmea	pygmy_marmoset	131	26
5	n5	cebus_capucinus	white_headed_capuchin	141	28
6	n6	mico_argentatus	silvery_marmoset	132	26
7	n7	saimiri_sciureus	common_squirrel_monkey	142	28
8	n8	aotus_nigriceps	black_headed_night_monkey	133	27
9	n9	trachypithecus_johnii	nilgiri_langur	132	26

In [8]:
```
labels_dict= {'n0':0, 'n1':1, 'n2':2, 'n3':3, 'n4':4, 'n5':5, 'n6':6, 'n7':7, 'n8':8, 'n9':9}

names_dict = dict(zip(labels_dict.values(), labels_info["Common Name"]))
print(names_dict)
```

```
{0: 'mantled_howler', 1: 'patas_monkey', 2: 'bald_uakari', 3: 'japanese_macaque', 4: 'pygmy_marmoset', 5: 'white_headed_capuc
hin', 6: 'silvery_marmoset', 7: 'common_squirrel_monkey', 8: 'black_headed_night_monkey', 9: 'nilgiri_langur'}
```

In [9]:

```python
train_df = []
for folder in os.listdir(training_data):
    imgs_path = training_data / folder

    imgs = sorted(imgs_path.glob('*.jpg'))

    for img_name in imgs:
        train_df.append((str(img_name), labels_dict[folder]))


train_df = pd.DataFrame(train_df, columns=['image', 'label'], index=None)
train_df = train_df.sample(frac=1.).reset_index(drop=True)
```

In [10]:

```python
valid_df = []
for folder in os.listdir(validation_data):
    imgs_path = validation_data / folder
    imgs = sorted(imgs_path.glob('*.jpg'))
    for img_name in imgs:
        valid_df.append((str(img_name), labels_dict[folder]))


valid_df = pd.DataFrame(valid_df, columns=['image', 'label'], index=None)
# shuffle the dataset
valid_df = valid_df.sample(frac=1.).reset_index(drop=True)
```

In [11]:

```python
print("Number of traininng samples: ", len(train_df))
print("Number of validation samples: ", len(valid_df))

print("\n",train_df.head(), "\n")
print("=================================================================\n")
print("\n", valid_df.head())
```

```
Number of traininng samples:  1097
Number of validation samples:  272

                         image  label
0  data\training\n0\n0134.jpg      0
1  data\training\n4\n4114.jpg      4
2  data\training\n4\n4059.jpg      4
3  data\training\n3\n3061.jpg      3
4  data\training\n0\n0150.jpg      0

=================================================================


                           image  label
0   data\validation\n2\n218.jpg      2
1  data\validation\n3\n3013.jpg      3
2  data\validation\n2\n2011.jpg      2
3   data\validation\n6\n608.jpg      6
4   data\validation\n6\n605.jpg      6
```

batch_size and some important constants

In [12]:
```
batch_size = 128
```

In [13]:
```
img_rows, img_cols, img_channels = 224,224,3
```

In [14]:
```
num_classes = 10
```

Creating the data generators to be used in the training stage

Augmentation pipeline

In [15]:
```
seq = iaa.OneOf([
    iaa.Fliplr(), # horizontal flips
```

```
    iaa.Affine(rotate=20), # roatation
    iaa.Multiply((1.2, 1.5))]) #random brightness
```

def data_generator(data, batch_size, is_validation_data=False):

In [16]:
```
def data_generator(data, batch_size, is_validation_data=False):

    n = len(data)
    nb_batches = int(np.ceil(n/batch_size))

    indices = np.arange(n)

    batch_data = np.zeros((batch_size, img_rows, img_cols, img_channels), dtype=np.float32)
    batch_labels = np.zeros((batch_size, num_classes), dtype=np.float32)

    while True:
        if not is_validation_data:
            np.random.shuffle(indices)

        for i in range(nb_batches):
            next_batch_indices = indices[i*batch_size:(i+1)*batch_size]

            for j, index in enumerate(next_batch_indices):
                img = cv2.imread(data.iloc[index]["image"])
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                label = data.iloc[index]["label"]

                if not is_validation_data:
                    img = seq.augment_image(img)

                img = cv2.resize(img, (img_rows, img_cols)).astype(np.float32)
                batch_data[j] = img
                batch_labels[j] = to_categorical(label,num_classes=num_classes)

            batch_data = preprocess_input(batch_data)
            yield batch_data, batch_labels
```

In [17]:
```
train_data_gen = data_generator(train_df, batch_size)

valid_data_gen = data_generator(valid_df, batch_size, is_validation_data=True)
```

Model

kernel_size

In [18]:
```
kernel_size = 2
```

Net Architecture

In [19]:
```
# https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D
# tf.keras.layers.Conv2D( filters, kernel_size, ...)

model = models.Sequential([
    layers.InputLayer( input_shape=(224, 224, 3) )
])

### 1 conv layer
model.add( layers.Conv2D(16, (kernel_size, kernel_size),
                        padding='same',
                        activation='relu') )
model.add( layers.MaxPooling2D((2, 2)) )

### 2 conv layer
model.add( layers.Conv2D(32, (kernel_size, kernel_size),
                        padding='same',
                        activation='relu') )
model.add( layers.MaxPooling2D((2, 2)) )

### 3 conv layer
model.add( layers.Conv2D(64, (kernel_size, kernel_size),
                        padding='same',
```

```
                        activation='relu') )
model.add( layers.MaxPooling2D((2, 2)) )

### 4 conv layer
model.add( layers.Conv2D(128, (kernel_size, kernel_size),
                        padding='same',
                        activation='relu') )
model.add( layers.MaxPooling2D((2, 2)) )

### 5 conv layer
model.add( layers.Conv2D(256, (kernel_size, kernel_size),
                        padding='same',
                        activation='relu') )
model.add( layers.MaxPooling2D((2, 2)) )

### 6 fully layers
model.add( layers.Flatten() )
model.add( layers.Dropout(0.2) )
model.add( layers.Dense(10, activation='softmax') )

# To correct some bug on input
model = Model(model.input, model.output)

optimizer = RMSprop(0.001)
model.compile(optimizer = optimizer,
              loss = 'categorical_crossentropy',
              metrics = ['accuracy'])

model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                Output Shape              Param #   
=================================================================
input_1 (InputLayer)        [(None, 224, 224, 3)]     0         
_________________________________________________________________
conv2d (Conv2D)             (None, 224, 224, 16)      208       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 112, 112, 16)     0         
_________________________________________________________________
conv2d_1 (Conv2D)           (None, 112, 112, 32)      2080      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 32)       0         
_________________________________________________________________
conv2d_2 (Conv2D)           (None, 56, 56, 64)        8256      
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 28, 28, 64)       0         
_________________________________________________________________
conv2d_3 (Conv2D)           (None, 28, 28, 128)       32896     
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 128)      0         
_________________________________________________________________
conv2d_4 (Conv2D)           (None, 14, 14, 256)       131328    
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 256)        0         
_________________________________________________________________
flatten (Flatten)           (None, 12544)             0         
_________________________________________________________________
dropout (Dropout)           (None, 12544)             0         
_________________________________________________________________
dense (Dense)               (None, 10)                125450    
=================================================================
Total params: 300,218
Trainable params: 300,218
Non-trainable params: 0
_________________________________________________________________
```

Setup to Training

EarlyStopping and ModelCheckpoint

In [20]:
```
early_stopping = EarlyStopping(patience=20, restore_best_weights=True)

model_checkpoint = ModelCheckpoint(filepath="model1", save_best_only=True)

num_train_steps = int(np.ceil(len(train_df)/batch_size))
num_valid_steps = int(np.ceil(len(valid_df)/batch_size))
```

Epochs

In [21]:
```
epochs=100
# epochs=5
```

Training

In [22]:
```
%%time

training_result = model.fit(train_data_gen,
                              epochs = epochs,
                              steps_per_epoch = num_train_steps,
                              validation_data = valid_data_gen,
                              validation_steps = num_valid_steps,
                              callbacks = [early_stopping, model_checkpoint])
```

```
Epoch 1/100
9/9 [==============================] - ETA: 0s - batch: 4.0000 - size: 128.0000 - loss: 18.9961 - accuracy: 0.1241
C:\Users\danie\anaconda3\envs\tf\lib\site-packages\keras\engine\training.py:2470: UserWarning: `Model.state_updates` will be removed in a future version. This property should not be used in TensorFlow 2.0, as `updates` are applied automatically.
  warnings.warn('`Model.state_updates` will be removed in a future version. '
```

```
INFO:tensorflow:Assets written to: model1\assets
9/9 [==============================] - 35s 4s/step - batch: 4.0000 - size: 128.0000 - loss: 18.9961 - accuracy: 0.1241 - val_
loss: 2.3242 - val_accuracy: 0.1328
Epoch 2/100
9/9 [==============================] - 25s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 2.3182 - accuracy: 0.1745 - val_l
oss: 2.3354 - val_accuracy: 0.1927
Epoch 3/100
9/9 [==============================] - ETA: 0s - batch: 4.0000 - size: 128.0000 - loss: 2.2077 - accuracy: 0.2240INFO:tensorf
low:Assets written to: model1\assets
9/9 [==============================] - 28s 4s/step - batch: 4.0000 - size: 128.0000 - loss: 2.2077 - accuracy: 0.2240 - val_l
oss: 2.1676 - val_accuracy: 0.2734
Epoch 4/100
9/9 [==============================] - 25s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 2.0785 - accuracy: 0.2682 - val_l
oss: 2.3840 - val_accuracy: 0.2344
Epoch 5/100
9/9 [==============================] - 27s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 2.1954 - accuracy: 0.2812 - val_l
oss: 2.9905 - val_accuracy: 0.1927
Epoch 6/100
9/9 [==============================] - ETA: 0s - batch: 4.0000 - size: 128.0000 - loss: 2.0846 - accuracy: 0.2457INFO:tensorf
low:Assets written to: model1\assets
9/9 [==============================] - 28s 4s/step - batch: 4.0000 - size: 128.0000 - loss: 2.0846 - accuracy: 0.2457 - val_l
oss: 2.0694 - val_accuracy: 0.2917
Epoch 7/100
9/9 [==============================] - 25s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 1.9951 - accuracy: 0.3394 - val_l
oss: 2.1789 - val_accuracy: 0.2318
Epoch 8/100
9/9 [==============================] - 27s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 1.8026 - accuracy: 0.3872 - val_l
oss: 2.0892 - val_accuracy: 0.3151
Epoch 9/100
9/9 [==============================] - ETA: 0s - batch: 4.0000 - size: 128.0000 - loss: 1.9085 - accuracy: 0.3767INFO:tensorf
low:Assets written to: model1\assets
9/9 [==============================] - 28s 4s/step - batch: 4.0000 - size: 128.0000 - loss: 1.9085 - accuracy: 0.3767 - val_l
oss: 1.8041 - val_accuracy: 0.3620
Epoch 10/100
9/9 [==============================] - 25s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 1.7984 - accuracy: 0.4115 - val_l
oss: 2.0721 - val_accuracy: 0.2891
Epoch 11/100
9/9 [==============================] - ETA: 0s - batch: 4.0000 - size: 128.0000 - loss: 1.5859 - accuracy: 0.4653INFO:tensorf
low:Assets written to: model1\assets
9/9 [==============================] - 28s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 1.5859 - accuracy: 0.4653 - val_l
```

```
oss: 1.6592 - val_accuracy: 0.3958
Epoch 12/100
9/9 [==============================] - 25s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 1.7859 - accuracy: 0.4123 - val_l
oss: 1.8186 - val_accuracy: 0.3750
Epoch 13/100
9/9 [==============================] - 26s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 1.4261 - accuracy: 0.5156 - val_l
oss: 2.2413 - val_accuracy: 0.3125
Epoch 14/100
9/9 [==============================] - ETA: 0s - batch: 4.0000 - size: 128.0000 - loss: 1.5471 - accuracy: 0.4714INFO:tensorf
low:Assets written to: model1\assets
9/9 [==============================] - 28s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 1.5471 - accuracy: 0.4714 - val_l
oss: 1.6323 - val_accuracy: 0.4401
Epoch 15/100
9/9 [==============================] - 25s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 1.5948 - accuracy: 0.4792 - val_l
oss: 1.6479 - val_accuracy: 0.4609
Epoch 16/100
9/9 [==============================] - 26s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 1.3585 - accuracy: 0.5694 - val_l
oss: 2.5646 - val_accuracy: 0.3594
Epoch 17/100
9/9 [==============================] - 26s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 1.3422 - accuracy: 0.5703 - val_l
oss: 1.7720 - val_accuracy: 0.4323
Epoch 18/100
9/9 [==============================] - 26s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 1.1251 - accuracy: 0.6155 - val_l
oss: 1.6583 - val_accuracy: 0.4844
Epoch 19/100
9/9 [==============================] - ETA: 0s - batch: 4.0000 - size: 128.0000 - loss: 1.2682 - accuracy: 0.5946INFO:tensorf
low:Assets written to: model1\assets
9/9 [==============================] - 28s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 1.2682 - accuracy: 0.5946 - val_l
oss: 1.6073 - val_accuracy: 0.4974
Epoch 20/100
9/9 [==============================] - ETA: 0s - batch: 4.0000 - size: 128.0000 - loss: 1.0960 - accuracy: 0.6172INFO:tensorf
low:Assets written to: model1\assets
9/9 [==============================] - 27s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 1.0960 - accuracy: 0.6172 - val_l
oss: 1.3013 - val_accuracy: 0.5208
Epoch 21/100
9/9 [==============================] - 25s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 1.1095 - accuracy: 0.6510 - val_l
oss: 2.5760 - val_accuracy: 0.3724
Epoch 22/100
9/9 [==============================] - 26s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 1.1305 - accuracy: 0.6380 - val_l
oss: 1.4293 - val_accuracy: 0.5286
```

```
Epoch 23/100
9/9 [==============================] - 27s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 1.0323 - accuracy: 0.6745 - val_l
oss: 1.5111 - val_accuracy: 0.4974
Epoch 24/100
9/9 [==============================] - ETA: 0s - batch: 4.0000 - size: 128.0000 - loss: 1.0636 - accuracy: 0.6493INFO:tensorf
low:Assets written to: model1\assets
9/9 [==============================] - 28s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 1.0636 - accuracy: 0.6493 - val_l
oss: 1.1986 - val_accuracy: 0.6016
Epoch 25/100
9/9 [==============================] - 25s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 0.8611 - accuracy: 0.7188 - val_l
oss: 1.9408 - val_accuracy: 0.4349
Epoch 26/100
9/9 [==============================] - 26s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 0.8009 - accuracy: 0.7543 - val_l
oss: 1.9984 - val_accuracy: 0.5078
Epoch 27/100
9/9 [==============================] - 26s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 0.8086 - accuracy: 0.7413 - val_l
oss: 1.3599 - val_accuracy: 0.5495
Epoch 28/100
9/9 [==============================] - 26s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 0.7049 - accuracy: 0.7604 - val_l
oss: 1.3455 - val_accuracy: 0.6094
Epoch 29/100
9/9 [==============================] - 27s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 0.7514 - accuracy: 0.7517 - val_l
oss: 1.2958 - val_accuracy: 0.5651
Epoch 30/100
9/9 [==============================] - 26s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 0.8114 - accuracy: 0.7361 - val_l
oss: 1.3288 - val_accuracy: 0.5781
Epoch 31/100
9/9 [==============================] - 26s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 0.6640 - accuracy: 0.8090 - val_l
oss: 1.8121 - val_accuracy: 0.4453
Epoch 32/100
9/9 [==============================] - 26s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 0.8072 - accuracy: 0.7509 - val_l
oss: 1.3066 - val_accuracy: 0.6016
Epoch 33/100
9/9 [==============================] - 26s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 0.6774 - accuracy: 0.7812 - val_l
oss: 1.2091 - val_accuracy: 0.5885
Epoch 34/100
9/9 [==============================] - 26s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 0.4977 - accuracy: 0.8611 - val_l
oss: 1.8419 - val_accuracy: 0.4740
Epoch 35/100
9/9 [==============================] - 26s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 0.8582 - accuracy: 0.7361 - val_l
```

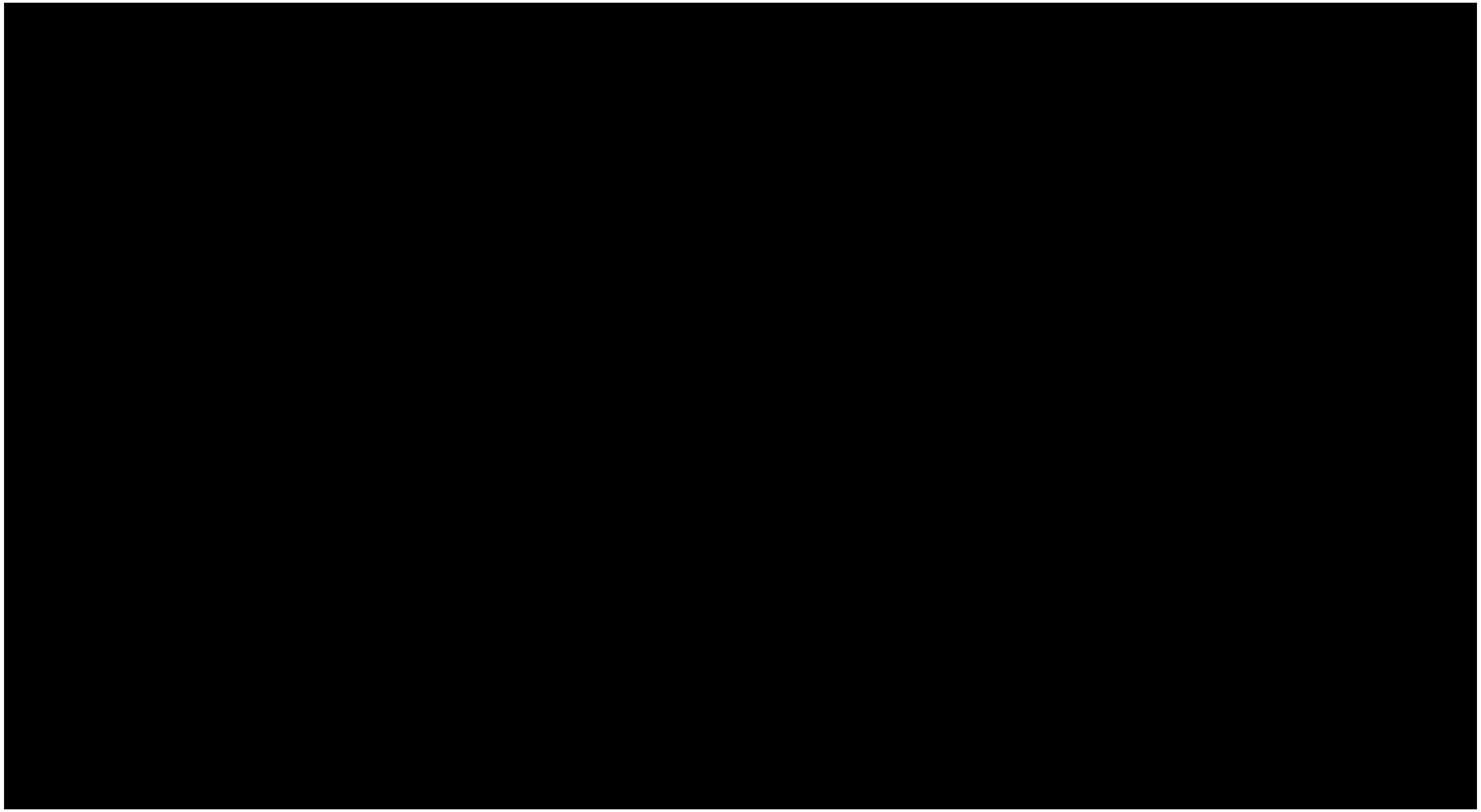
```
oss: 1.2836 - val_accuracy: 0.6276
Epoch 36/100
9/9 [==============================] - 26s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 0.5160 - accuracy: 0.8429 - val_l
oss: 2.8456 - val_accuracy: 0.4922
Epoch 37/100
9/9 [==============================] - 26s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 0.5767 - accuracy: 0.8203 - val_l
oss: 1.3547 - val_accuracy: 0.5651
Epoch 38/100
9/9 [==============================] - 27s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 0.8158 - accuracy: 0.7691 - val_l
oss: 1.3265 - val_accuracy: 0.5938
Epoch 39/100
9/9 [==============================] - 26s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 0.5158 - accuracy: 0.8524 - val_l
oss: 2.5891 - val_accuracy: 0.4714
Epoch 40/100
9/9 [==============================] - 26s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 0.4946 - accuracy: 0.8611 - val_l
oss: 1.8541 - val_accuracy: 0.4740
Epoch 41/100
9/9 [==============================] - 27s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 0.4843 - accuracy: 0.8464 - val_l
oss: 1.5196 - val_accuracy: 0.5547
Epoch 42/100
9/9 [==============================] - 26s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 0.3488 - accuracy: 0.8967 - val_l
oss: 1.3932 - val_accuracy: 0.6328
Epoch 43/100
9/9 [==============================] - 26s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 0.4628 - accuracy: 0.8689 - val_l
oss: 1.5748 - val_accuracy: 0.5547
Epoch 44/100
9/9 [==============================] - 26s 3s/step - batch: 4.0000 - size: 128.0000 - loss: 0.4755 - accuracy: 0.8707 - val_l
oss: 1.7256 - val_accuracy: 0.5911
CPU times: total: 39min 1s
Wall time: 19min 25s
```

Results

In [23]:
```
training_result.history.keys()
```

Out[23]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [24]:
```
train_acc = training_result.history['accuracy']
valid_acc = training_result.history['val_accuracy']

train_loss = training_result.history['loss']
valid_loss = training_result.history['val_loss']

xvalues = np.arange(len(train_acc))

f,ax = plt.subplots(1,2, figsize=(10,5))
ax[0].plot(xvalues, train_loss)
ax[0].plot(xvalues, valid_loss)
ax[0].set_title("Loss curve")
ax[0].set_xlabel("Epoch")
ax[0].set_ylabel("loss")
ax[0].legend(['train', 'validation'])

ax[1].plot(xvalues, train_acc)
ax[1].plot(xvalues, valid_acc)
ax[1].set_title("Accuracy")
ax[1].set_xlabel("Epoch")
ax[1].set_ylabel("accuracy")
ax[1].legend(['train', 'validation'])

plt.show()
```



In [26]:
```
valid_loss, valid_acc = model.evaluate_generator(valid_data_gen, steps=num_valid_steps)
print(f"Final validation accuracy: {valid_acc*100:.2f}%")
```

```
Final validation accuracy: 58.33%
```

In [27]:
```
outputs = [layer.output for layer in model.layers[1:18]]

vis_model = Model(model.input, outputs)

for layer in vis_model.layers:
    layer.trainable = False

# vis_model.summary()
```

In [28]:
```
layer_names = []
for layer in outputs:
    layer_names.append(layer.name.split("/")[0])


print("Layers going to be used for visualization: ")
print(layer_names)
```

```
Layers going to be used for visualization: 
['conv2d', 'max_pooling2d', 'conv2d_1', 'max_pooling2d_1', 'conv2d_2', 'max_pooling2d_2', 'conv2d_3', 'max_pooling2d_3', 'con
v2d_4', 'max_pooling2d_4', 'flatten', 'dropout', 'dense']
```

In [29]:
```
print( f'layer_names [before]: {layer_names}' )

layer_names_temp = layer_names
layer_names = list()
for layer in layer_names_temp:

    if 'conv' in layer:
        # print(layer)
        layer_names.append( layer )

print( '==================================================================================' )
print( f'layer_names [after]: {layer_names}' )
```

```
layer_names [before]: ['conv2d', 'max_pooling2d', 'conv2d_1', 'max_pooling2d_1', 'conv2d_2', 'max_pooling2d_2', 'conv2d_3',
'max_pooling2d_3', 'conv2d_4', 'max_pooling2d_4', 'flatten', 'dropout', 'dense']
==================================================================================
layer_names [after]: ['conv2d', 'conv2d_1', 'conv2d_2', 'conv2d_3', 'conv2d_4']
```

In [30]:
```
def get_CAM(processed_image, predicted_label):

    predicted_output = model.output[:, predicted_label]

    last_conv_layer = model.get_layer(layer_names[-1])

    # get the gradients wrt to the last conv layer
    grads = K.gradients(predicted_output, last_conv_layer.output)[0]

    # take mean gradient per feature map
    grads = K.mean(grads, axis=(0,1,2)) # GAP - Global Average Pooling
```

```
    # Define a function that generates the values for the output and gradients
    evaluation_function = K.function([model.input], [grads, last_conv_layer.output[0]])

    # get the values
    grads_values, conv_ouput_values = evaluation_function([processed_image])

    # CAM - Class Activation Map
    # iterate over each feature map in your conv output and multiply
    # the gradient values with the conv output values. This gives an
    # indication of "how important a feature is"
    # for i in features in the last conv layer.
    for i in range(conv_ouput_values.shape[2]):
        conv_ouput_values[:,:,i] *= grads_values[i]

    heatmap = np.mean(conv_ouput_values, axis=-1)
    heatmap = np.maximum(heatmap, 0)
    heatmap /= heatmap.max()

    return heatmap
```

Examples

In [34]:
```
for index in range(0, 10):

    sample_image = cv2.imread(valid_df.iloc[index]['image'])
    sample_image = cv2.cvtColor(sample_image, cv2.COLOR_BGR2RGB)
    sample_image = cv2.resize(sample_image, (img_rows, img_cols))
    sample_label = valid_df.iloc[index]["label"]

    sample_image_processed = np.expand_dims(sample_image, axis=0)
    sample_image_processed = preprocess_input(sample_image_processed)

    pred_label = np.argmax(model.predict(sample_image_processed), axis=-1)[0]

    heatmap = get_CAM(sample_image_processed, pred_label)
    heatmap = cv2.resize(heatmap, (sample_image.shape[0], sample_image.shape[1]))
    heatmap = heatmap *255
```

```
    heatmap = np.clip(heatmap, 0, 255).astype(np.uint8)
    heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)
    super_imposed_image = heatmap * 0.5 + sample_image
    super_imposed_image = np.clip(super_imposed_image, 0,255).astype(np.uint8)

    fontsize = 10
    fig, axes = plt.subplots( 1, 3, figsize=( 8, 8 ) )
    axes[0].set_title( f'True label: {sample_label} \n Predicted label: {pred_label}',
                      fontsize = fontsize )
    axes[0].axis('off')
    axes[0].imshow( sample_image )

    axes[1].set_title( f'Class Activation Map',
                      fontsize = fontsize )
    axes[1].axis('off')
    axes[1].imshow( heatmap )

    axes[2].set_title( f'Activation Map Superimposed',
                      fontsize = fontsize )
    axes[2].axis('off')
    axes[2].imshow( super_imposed_image )
    plt.show()

    # # Plot just CAM of the layer
    # plt.figure( figsize=(2, 2) )
    # plt.title( f'Class Activation Map - Layer: {layer}' )
    # plt.imshow( heatmap )
    # plt.show()
```