Deep Learning HA

December 12, 2023

1 Importing neccesary libraries

[1]:
```
import gdown
import zipfile
import pandas as pd
import random
import warnings

warnings.filterwarnings('ignore')

from PIL import Image
from IPython.display import display
import matplotlib.pyplot as plt

import tensorflow as tf
```

```
WARNING:tensorflow:From C:\Users\malir\anaconda3\lib\site-
packages\keras\src\losses.py:2976: The name
tf.losses.sparse_softmax_cross_entropy is deprecated. Please use
tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

2 Downloading and storing images and data sets

2.1 Images

[2]:
```
image_url = 'https://drive.google.com/uc?id=18c43SiRTrJYjUN0cPoynA39NWx__bJDa'
image_zip = 'images.zip'

gdown.download(image_url, image_zip, quiet=False)

extracted = 'images'
with zipfile.ZipFile(image_zip, 'r') as zip_ref:
    zip_ref.extractall(extracted)
```

```
Downloading…
From: https://drive.google.com/uc?id=18c43SiRTrJYjUN0cPoynA39NWx__bJDa
To: C:\Users\malir\images.zip
```

```
100%|
 | 11.5M/11.5M [00:02<00:00, 5.51MB/s]
```

2.2 Dataset

[3]:
```python
target_url = 'https://drive.google.com/uc?id=1_lwltpA4uhAchTHy7M6fe3tM_eCRGqjU'
target_csv = 'target_data.csv'

gdown.download(target_url, target_csv, quiet=False)

df = pd.read_csv(target_csv,sep='\t')
```

```
Downloading…
From: https://drive.google.com/uc?id=1_lwltpA4uhAchTHy7M6fe3tM_eCRGqjU
To: C:\Users\malir\target_data.csv
100%|
  | 134k/134k [00:00<00:00, 2.25MB/s]
```

3 Make a list of images names to match with the data set and find the corresponding ones

[4]:
```python
import os
from os import listdir

pic_dir = 'images/imgs'

pic_names = []
for pic in os.listdir(pic_dir):
    pic_names.append(pic)

print(len(pic_names))
```

```
1764
```

4 Check wether all images have the same resolution or not

[5]:
```python
resolutions = {}
for i in pic_names:
    img = Image.open(pic_dir+f'/{i}')
    wid, hgt = img.size
    if (wid, hgt) not in resolutions:
        resolutions[(wid, hgt)] = 0
    resolutions[(wid, hgt)] += 1


resolutions
```

[5]: {(432, 432): 1764}

4.1 Storing the resolution in a variable in case we need that

[6]:
```
img_res = next(iter(resolutions.keys()))
```

5 Add '.png' to the ad_clicked column and changing the data type to string so that we can check the list of images with that, and get rid of the ones that are not present amongst our images

[7]:
```
df['log_id'] = df['log_id'].astype('str') + '.png'
df
```

[7]:

	user_id	ad_clicked	attention	log_id
0	5npsk114ba8hfbj4jr3lt8jhf5	0	4	20181002033126.png
1	5o9js8slc8rg2a8mo5p3r93qm0	1	5	20181001211223.png
2	pi17qjfqmnhpsiahbumcsdq0r6	0	4	20181001170952.png
3	3rptg9g7l83imkbdsu2miignv7	0	1	20181001140754.png
4	049onniafv6fe4e6q42k6nq1n2	0	1	20181001132434.png
…	…	…	…	…
2904	2jbfmshmhsji4smrgph018k410	0	2	20170203232414.png
2905	p1tt6ehhpcihelra9j558acgv7	0	4	20170131193748.png
2906	tl1hfafsot8s5qud19bkij68f7	0	2	20170106152837.png
2907	lvmrfennsqgn49ndepfnl68ok4	0	4	20170102171535.png
2908	bsqgffkob06hgsb6r0csjk4gv6	0	4	20161227191740.png

[2909 rows x 4 columns]

5.1 Dropping the ones that are not in our list

[8]:
```
df = df.sort_values(by=['log_id'])

for i in df['log_id'].values:
    if i not in pic_names:
        df.drop(df[df['log_id'] == i].index,inplace=True)

df.reset_index(inplace=True,drop=True)
```

[9]:
```
df
```

[9]:

	user_id	ad_clicked	attention	log_id
0	2fq3fhu6r693jl2sdb8fhbd190	1	5	20161214224444.png
1	mh0d0jr0oo98kriolus0ml0591	0	2	20161215173310.png
2	hp3hqt4ifrb3q4f8vgo8e98e65	0	4	20161217041101.png
3	9k7e1d73n5reh06p705cgp2ke7	0	2	20161218020519.png
4	j25njapljcob43qrg1cfnsg452	0	4	20161219223751.png

```
...                                ...          ...        ...                  ...
1759  r79vp0b4cm4aahos39seof8ei1            0          5  20180627155928.png
1760  5rhdto3kle22lctorjr4ouqvb4            0          5  20180627181020.png
1761  58npdb91133qlb8bfoejokope3            1          4  20180627205851.png
1762  uhlg8fr2vaejph65n7crkq4ln1            0          4  20180628045013.png
1763  rhth03rhqi73p4hm3thdqrfqs3            0          4  20180628045518.png

[1764 rows x 4 columns]
```

[10]:
```
df['ad_clicked'].value_counts()
```

[10]:
```
0    1270
1     494
Name: ad_clicked, dtype: int64
```

We can clearly observe that the 72% of our data belong to 0 label and 28% belong to 1 label, which shows an imbalance in our labels

5.2 Check if the ad_clicked column is 100 % similar to the images list

[11]:
```
print(all(pic_names == df['log_id'].values))
```

```
True
```

6 Converting the images to NumPy arrays

[12]:
```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

pic_dir = 'images/imgs'
labels = df['ad_clicked']

# First i defiend a function to read and process the image using cv2 library
# The installation syntax using pip -> pip install numpy matplotlib␣
↪opencv-python
# We will resize images from 432*432 to 128*128
def preprocess_image(pic_dir, target_size=(128, 128)):
    image = cv2.imread(pic_dir)

    # This step is necessary because in the documentation it's been told that␣
↪OpenCV reads images in BGR format by default
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Resizing to 128*128 to make the calculations and reading easier
```

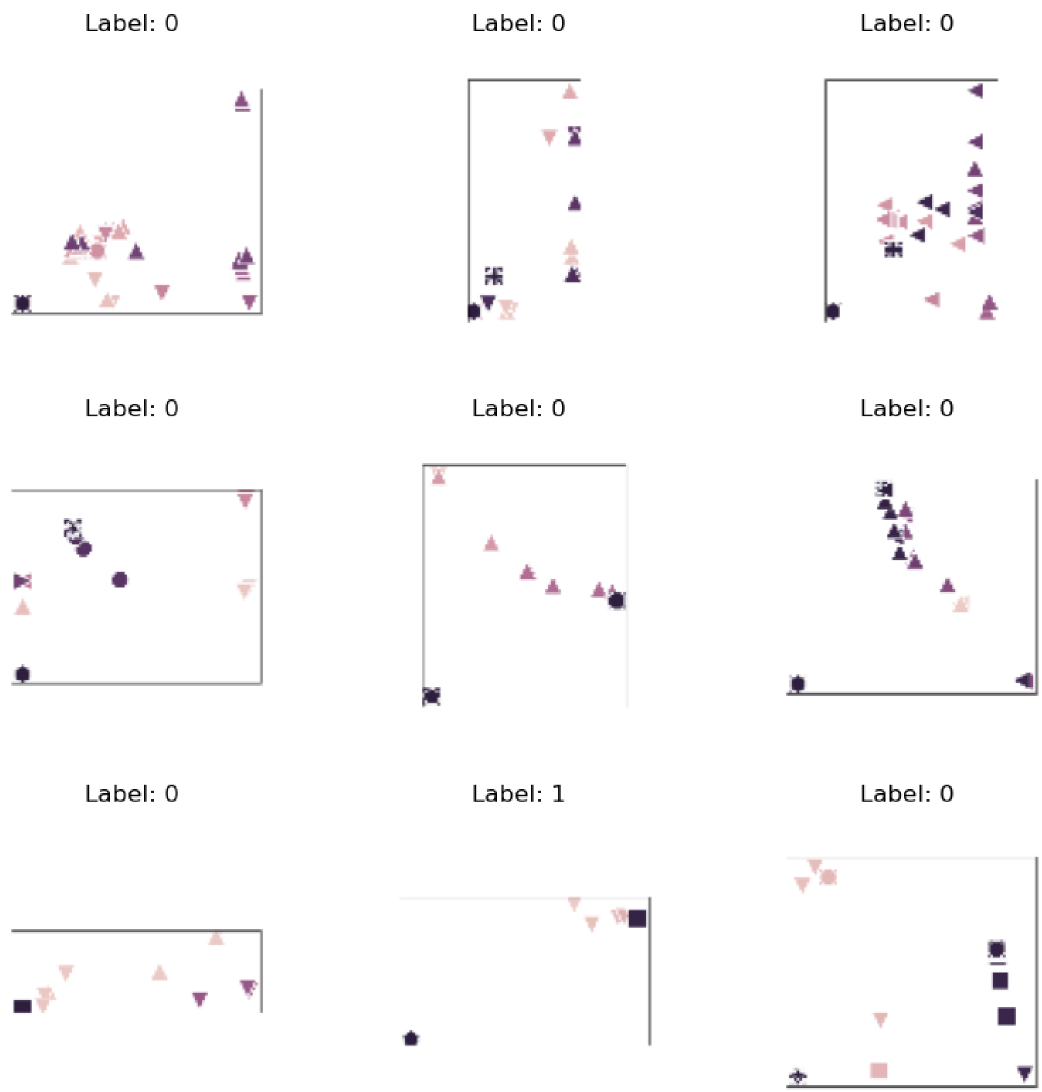
```
    image = cv2.resize(image, target_size)

    # Most important part, normalizing pixel values to be in range of (0,1)
    image = image / 255.0
    return image

# Images to NumPy array
image_files = [os.path.join(pic_dir, filename) for filename in os.
 ↪listdir(pic_dir)]

images = [preprocess_image(file_path) for file_path in image_files]
labels = labels.values

# Split the data into train,test and validation
train_images, test_val_images, train_labels, test_val_labels = train_test_split(
    images, labels, test_size=0.3, random_state=42
)

valid_images, test_images, valid_labels, test_labels = train_test_split(
    test_val_images, test_val_labels, test_size=0.5, random_state=42
)


# Convert to NumPy arrays
train_images = np.array(train_images)
valid_images = np.array(valid_images)
test_images = np.array(test_images)
```

[13]:
```
print(f"Train Images Shape -> {train_images.shape}")
print(f"Train Labels Shape -> {train_labels.shape}")
print(f"Valid Images Shape -> {valid_images.shape}")
print(f"Valid Labels Shape -> {valid_labels.shape}")
print(f"Test Images Shape -> {test_images.shape}")
print(f"Test Labels Shape -> {test_labels.shape}")
```

```
Train Images Shape -> (1234, 128, 128, 3)
Train Labels Shape -> (1234,)
Valid Images Shape -> (265, 128, 128, 3)
Valid Labels Shape -> (265,)
Test Images Shape -> (265, 128, 128, 3)
Test Labels Shape -> (265,)
```

7 Plotting randomly some of our images

[14]:
```python
import matplotlib.pyplot as plt
import random

num_images = len(train_images)

# Generating 9 different random numbers to randomly illustrate some of images
random_indices = random.sample(range(num_images), 9)

plt.figure(figsize=(10, 10))
for i, idx in enumerate(random_indices, 1):
    plt.subplot(3, 3, i)
    plt.imshow(train_images[idx])
    plt.title(f"Label: {train_labels[idx]}")
    plt.axis("off")

plt.show()
```



8 Implementing the Neural Network

8.1 First step is to import the libraries that we need, i will use the sequnetial api of keras library

[15]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras import backend as K
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.regularizers import l2
```

First implementation is more based on gut feelings for me because this is the first time that i am handling the image processing, so i will use three convolutional layers and two dense layers, of course the last layer is our output and it will only contain one neuron with sigmoid activation, the out put of such layer can only be 0 and 1 which is suitable for our task which is a binary classification.

[16]:
```
# First create the empty model
model = Sequential()

# Very important to clear the session to avoid mixing up with and being␣
↪affetced by previous computations
tf.keras.backend.clear_session()

# First convolutional layers gets the input which is resolution and channels in␣
↪our case -> (128,128,3)
# Using MaxPooling => reduce the spatial dimensions of the input,while␣
↪retaining the most important information
model.add(Conv2D(32, 3, kernel_initializer='normal', activation='relu',␣
↪input_shape=train_images[0].shape))
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Conv2D(32, 3, kernel_initializer='normal', activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Conv2D(64, 3, kernel_initializer='normal', activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Flatten())

# Using drop which out is another regularization technique
# dropout rate = 0.45 ==> 45% of the input units will be randomly set to zero␣
↪(Drop)
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.45))

model.add(Dense(1, activation='sigmoid'))
```

```
WARNING:tensorflow:From C:\Users\malir\anaconda3\lib\site-
packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated.
Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\malir\anaconda3\lib\site-
packages\keras\src\layers\pooling\max_pooling2d.py:161: The name tf.nn.max_pool
is deprecated. Please use tf.nn.max_pool2d instead.
```

[17]:
```
model.summary()
```

```
Model: "sequential"
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 32)      896

 max_pooling2d (MaxPooling2  (None, 62, 62, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 60, 60, 32)        9248

 max_pooling2d_1 (MaxPoolin  (None, 29, 29, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 27, 27, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 13, 13, 64)        0
 g2D)

 flatten (Flatten)           (None, 10816)             0

 dense (Dense)               (None, 64)                692288

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 720993 (2.75 MB)
Trainable params: 720993 (2.75 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Then we compile the model with apropriate loss and optimizer, ofcourse we can decide about the loss function, becase it's a binary classification we might go with the binary crossentropy but optimizer is a hyper parameter and needs to be tuned. After compiling, finally we fit the model on our train data and see how it performs on the validation data, the loss function and accuracy are important metrics here since they show how model perform in terms of under fitting and over fitting, wether the model learns or not.

[18]:
```
model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.0001),
 ↪metrics=['accuracy'])
results = model.fit(train_images,
             train_labels,
             epochs=12,
             batch_size=100,
             validation_data=(valid_images, valid_labels))
```

```
Epoch 1/12
```

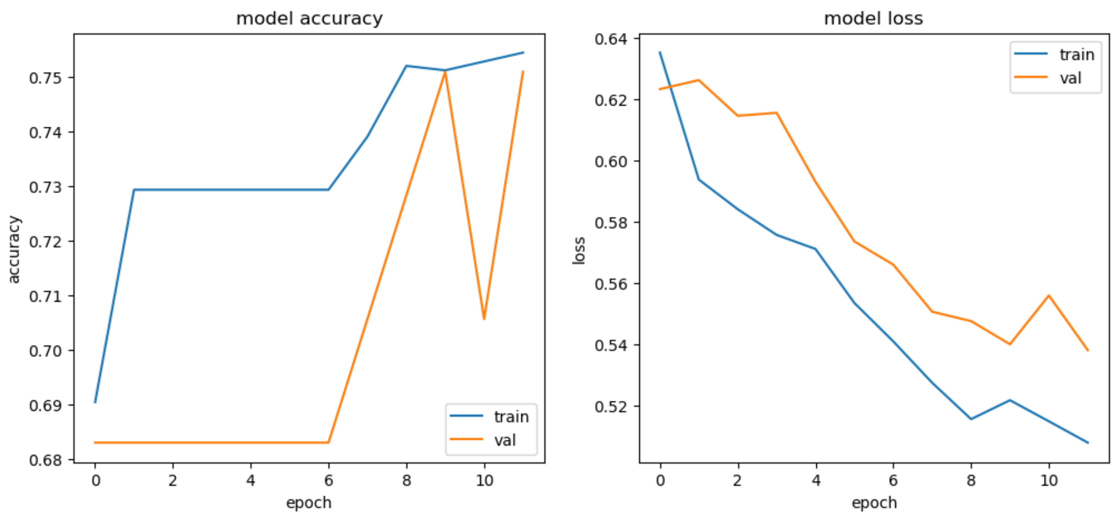
```
WARNING:tensorflow:From C:\Users\malir\anaconda3\lib\site-
packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue
is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\malir\anaconda3\lib\site-
packages\keras\src\engine\base_layer_utils.py:384: The name
tf.executing_eagerly_outside_functions is deprecated. Please use
tf.compat.v1.executing_eagerly_outside_functions instead.

13/13 [==============================] - 6s 357ms/step - loss: 0.6352 -
accuracy: 0.6904 - val_loss: 0.6233 - val_accuracy: 0.6830
Epoch 2/12
13/13 [==============================] - 3s 268ms/step - loss: 0.5938 -
accuracy: 0.7293 - val_loss: 0.6262 - val_accuracy: 0.6830
Epoch 3/12
13/13 [==============================] - 4s 287ms/step - loss: 0.5841 -
accuracy: 0.7293 - val_loss: 0.6146 - val_accuracy: 0.6830
Epoch 4/12
13/13 [==============================] - 4s 283ms/step - loss: 0.5757 -
accuracy: 0.7293 - val_loss: 0.6156 - val_accuracy: 0.6830
Epoch 5/12
13/13 [==============================] - 3s 260ms/step - loss: 0.5711 -
accuracy: 0.7293 - val_loss: 0.5930 - val_accuracy: 0.6830
Epoch 6/12
13/13 [==============================] - 4s 276ms/step - loss: 0.5535 -
accuracy: 0.7293 - val_loss: 0.5735 - val_accuracy: 0.6830
Epoch 7/12
13/13 [==============================] - 4s 278ms/step - loss: 0.5409 -
accuracy: 0.7293 - val_loss: 0.5660 - val_accuracy: 0.6830
Epoch 8/12
13/13 [==============================] - 4s 275ms/step - loss: 0.5274 -
accuracy: 0.7391 - val_loss: 0.5506 - val_accuracy: 0.7057
Epoch 9/12
13/13 [==============================] - 4s 283ms/step - loss: 0.5155 -
accuracy: 0.7520 - val_loss: 0.5475 - val_accuracy: 0.7283
Epoch 10/12
13/13 [==============================] - 4s 295ms/step - loss: 0.5217 -
accuracy: 0.7512 - val_loss: 0.5400 - val_accuracy: 0.7509
Epoch 11/12
13/13 [==============================] - 4s 298ms/step - loss: 0.5148 -
accuracy: 0.7528 - val_loss: 0.5559 - val_accuracy: 0.7057
Epoch 12/12
13/13 [==============================] - 4s 270ms/step - loss: 0.5079 -
accuracy: 0.7545 - val_loss: 0.5380 - val_accuracy: 0.7509
```

Function for plotting loss and accuracy for train and validation data (Copied from one of the notebooks :D)

[19]:
```
def plot_results(results):
  # summarize history for accuracy
  plt.figure(figsize = (12,5))
  plt.subplot(121)
  plt.plot(results.history['accuracy'])
  plt.plot(results.history['val_accuracy'])
  plt.title('model accuracy')
  plt.ylabel('accuracy')
  plt.xlabel('epoch')
  plt.legend(['train', 'val'], loc='lower right')

  # summarize history for loss
  plt.subplot(122)
  plt.plot(results.history['loss'])
  plt.plot(results.history['val_loss'])
  plt.title('model loss')
  plt.ylabel('loss')
  plt.xlabel('epoch')
  plt.legend(['train', 'val'], loc='upper right')

  max_loss = np.max(results.history['loss'])
  min_loss = np.min(results.history['loss'])
  print("Maximum Loss : {:.4f}".format(max_loss))
  print("")
  print("Minimum Loss : {:.4f}".format(min_loss))
  print("")
  print("Loss difference : {:.4f}".format((max_loss - min_loss)))
```

[20]:
```
plot_results(results)
```

```
Maximum Loss : 0.6352

Minimum Loss : 0.5079

Loss difference : 0.1274
```



When the validation loss is higher than the training loss, it typically indicates that the model is facing overfitting, since the training accuracy is relatively high (74.31%), and the validation accuracy is lower, it shows that the model is overfitting. The model is likely learning specific patterns in the training data that do not generalize well to new data (validation data). To prevent overfitting we apply regularization techniques to tackle with large weights and i think L2 is a good choice in this case because it adds the sum of squared weights to the loss function, promoting smaller but non-zero weights and preventing extreme values, i will also increase the dropout rate to 0.5 and increase number of epochs and decrease the batch size, it will allow the model to learn from each individual example but takes longer to train.

[21]:
```
# First create the empty model
model = Sequential()

# Very important to clear the session to avoid mixing up with and being␣
↪affetced by previous computations
tf.keras.backend.clear_session()

# First convolutional layers gets the input which is resolution and channels in␣
↪our case -> (128,128,3)
# Using MaxPooling => reduce the spatial dimensions of the input,while␣
↪retaining the most important information
model.add(Conv2D(32, 3, kernel_initializer='normal', activation='relu',␣
↪input_shape=train_images[0].shape))
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Conv2D(32, 3, kernel_initializer='normal', activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Conv2D(64, 3, kernel_initializer='normal', activation='relu'))
```

```
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Flatten())

# Using drop which out is another regularization technique
# dropout rate = 0.45 ==> 45% of the input units will be randomly set to zero␣
↪(Drop)
# Adding the kernel regulizer,l2 and increasing dropout rate
model.add(Dense(64, activation='relu',kernel_regularizer=l2(0.01)))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))
```

[22]:
```
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 32)      896

 max_pooling2d (MaxPooling2  (None, 62, 62, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 60, 60, 32)        9248

 max_pooling2d_1 (MaxPoolin  (None, 29, 29, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 27, 27, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 13, 13, 64)        0
 g2D)

 flatten (Flatten)           (None, 10816)             0

 dense (Dense)               (None, 64)                692288

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 720993 (2.75 MB)
Trainable params: 720993 (2.75 MB)
Non-trainable params: 0 (0.00 Byte)

_________________________________________________________________
```

[23]:
```
model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.0001),␣
↪metrics=['accuracy'])
results = model.fit(train_images,
             train_labels,
             epochs=40,
             batch_size=30,
             validation_data=(valid_images, valid_labels))
```

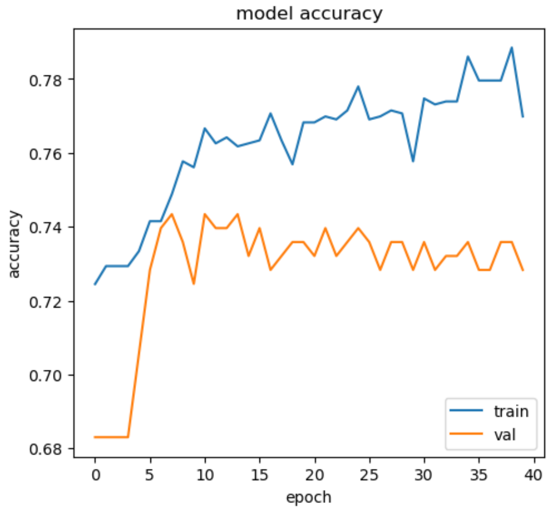
```
Epoch 1/40
42/42 [==============================] - 7s 119ms/step - loss: 1.6445 -
accuracy: 0.7245 - val_loss: 1.4557 - val_accuracy: 0.6830
Epoch 2/40
42/42 [==============================] - 6s 132ms/step - loss: 1.2651 -
accuracy: 0.7293 - val_loss: 1.1520 - val_accuracy: 0.6830
Epoch 3/40
42/42 [==============================] - 6s 137ms/step - loss: 1.0171 -
accuracy: 0.7293 - val_loss: 0.9664 - val_accuracy: 0.6830
Epoch 4/40
42/42 [==============================] - 5s 122ms/step - loss: 0.8634 -
accuracy: 0.7293 - val_loss: 0.8157 - val_accuracy: 0.6830
Epoch 5/40
42/42 [==============================] - 5s 129ms/step - loss: 0.7613 -
accuracy: 0.7334 - val_loss: 0.7437 - val_accuracy: 0.7057
Epoch 6/40
42/42 [==============================] - 4s 99ms/step - loss: 0.6943 - accuracy:
0.7415 - val_loss: 0.6963 - val_accuracy: 0.7283
Epoch 7/40
42/42 [==============================] - 4s 93ms/step - loss: 0.6715 - accuracy:
0.7415 - val_loss: 0.6669 - val_accuracy: 0.7396
Epoch 8/40
42/42 [==============================] - 4s 97ms/step - loss: 0.6277 - accuracy:
0.7488 - val_loss: 0.6441 - val_accuracy: 0.7434
Epoch 9/40
42/42 [==============================] - 5s 110ms/step - loss: 0.6235 -
accuracy: 0.7577 - val_loss: 0.6416 - val_accuracy: 0.7358
Epoch 10/40
42/42 [==============================] - 4s 97ms/step - loss: 0.6079 - accuracy:
0.7561 - val_loss: 0.6372 - val_accuracy: 0.7245
Epoch 11/40
42/42 [==============================] - 5s 110ms/step - loss: 0.5921 -
accuracy: 0.7666 - val_loss: 0.6159 - val_accuracy: 0.7434
Epoch 12/40
42/42 [==============================] - 4s 97ms/step - loss: 0.5791 - accuracy:
0.7626 - val_loss: 0.6037 - val_accuracy: 0.7396
Epoch 13/40
42/42 [==============================] - 4s 87ms/step - loss: 0.5690 - accuracy:
0.7642 - val_loss: 0.6095 - val_accuracy: 0.7396
Epoch 14/40
```

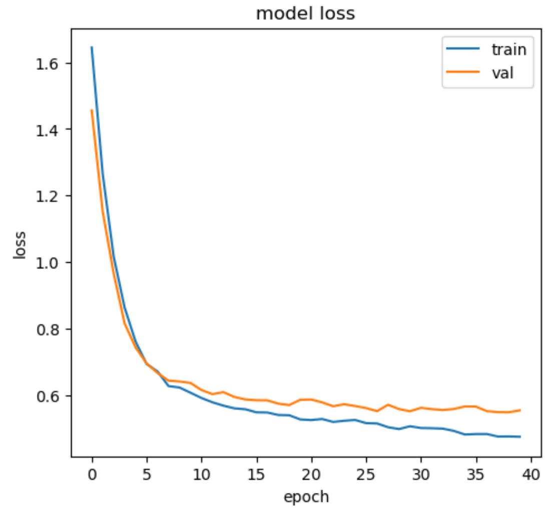
```
42/42 [==============================] - 4s 90ms/step - loss: 0.5609 - accuracy:
0.7618 - val_loss: 0.5952 - val_accuracy: 0.7434
Epoch 15/40
42/42 [==============================] - 4s 95ms/step - loss: 0.5582 - accuracy:
0.7626 - val_loss: 0.5876 - val_accuracy: 0.7321
Epoch 16/40
42/42 [==============================] - 4s 92ms/step - loss: 0.5491 - accuracy:
0.7634 - val_loss: 0.5853 - val_accuracy: 0.7396
Epoch 17/40
42/42 [==============================] - 4s 104ms/step - loss: 0.5485 -
accuracy: 0.7707 - val_loss: 0.5848 - val_accuracy: 0.7283
Epoch 18/40
42/42 [==============================] - 4s 91ms/step - loss: 0.5410 - accuracy:
0.7634 - val_loss: 0.5751 - val_accuracy: 0.7321
Epoch 19/40
42/42 [==============================] - 4s 105ms/step - loss: 0.5402 -
accuracy: 0.7569 - val_loss: 0.5708 - val_accuracy: 0.7358
Epoch 20/40
42/42 [==============================] - 4s 95ms/step - loss: 0.5276 - accuracy:
0.7682 - val_loss: 0.5866 - val_accuracy: 0.7358
Epoch 21/40
42/42 [==============================] - 5s 125ms/step - loss: 0.5258 -
accuracy: 0.7682 - val_loss: 0.5873 - val_accuracy: 0.7321
Epoch 22/40
42/42 [==============================] - 5s 115ms/step - loss: 0.5288 -
accuracy: 0.7699 - val_loss: 0.5789 - val_accuracy: 0.7396
Epoch 23/40
42/42 [==============================] - 5s 109ms/step - loss: 0.5201 -
accuracy: 0.7690 - val_loss: 0.5672 - val_accuracy: 0.7321
Epoch 24/40
42/42 [==============================] - 6s 135ms/step - loss: 0.5236 -
accuracy: 0.7715 - val_loss: 0.5732 - val_accuracy: 0.7358
Epoch 25/40
42/42 [==============================] - 6s 139ms/step - loss: 0.5261 -
accuracy: 0.7780 - val_loss: 0.5679 - val_accuracy: 0.7396
Epoch 26/40
42/42 [==============================] - 5s 120ms/step - loss: 0.5165 -
accuracy: 0.7690 - val_loss: 0.5618 - val_accuracy: 0.7358
Epoch 27/40
42/42 [==============================] - 6s 155ms/step - loss: 0.5155 -
accuracy: 0.7699 - val_loss: 0.5525 - val_accuracy: 0.7283
Epoch 28/40
42/42 [==============================] - 5s 113ms/step - loss: 0.5043 -
accuracy: 0.7715 - val_loss: 0.5716 - val_accuracy: 0.7358
Epoch 29/40
42/42 [==============================] - 4s 101ms/step - loss: 0.4988 -
accuracy: 0.7707 - val_loss: 0.5588 - val_accuracy: 0.7358
Epoch 30/40
```

```
42/42 [==============================] - 4s 98ms/step - loss: 0.5071 - accuracy:
0.7577 - val_loss: 0.5522 - val_accuracy: 0.7283
Epoch 31/40
42/42 [==============================] - 4s 103ms/step - loss: 0.5019 -
accuracy: 0.7747 - val_loss: 0.5625 - val_accuracy: 0.7358
Epoch 32/40
42/42 [==============================] - 4s 95ms/step - loss: 0.5013 - accuracy:
0.7731 - val_loss: 0.5585 - val_accuracy: 0.7283
Epoch 33/40
42/42 [==============================] - 4s 93ms/step - loss: 0.5000 - accuracy:
0.7739 - val_loss: 0.5559 - val_accuracy: 0.7321
Epoch 34/40
42/42 [==============================] - 4s 97ms/step - loss: 0.4932 - accuracy:
0.7739 - val_loss: 0.5590 - val_accuracy: 0.7321
Epoch 35/40
42/42 [==============================] - 4s 91ms/step - loss: 0.4824 - accuracy:
0.7861 - val_loss: 0.5666 - val_accuracy: 0.7358
Epoch 36/40
42/42 [==============================] - 4s 90ms/step - loss: 0.4838 - accuracy:
0.7796 - val_loss: 0.5662 - val_accuracy: 0.7283
Epoch 37/40
42/42 [==============================] - 4s 86ms/step - loss: 0.4838 - accuracy:
0.7796 - val_loss: 0.5525 - val_accuracy: 0.7283
Epoch 38/40
42/42 [==============================] - 4s 87ms/step - loss: 0.4763 - accuracy:
0.7796 - val_loss: 0.5496 - val_accuracy: 0.7358
Epoch 39/40
42/42 [==============================] - 4s 88ms/step - loss: 0.4766 - accuracy:
0.7885 - val_loss: 0.5491 - val_accuracy: 0.7358
Epoch 40/40
42/42 [==============================] - 4s 85ms/step - loss: 0.4756 - accuracy:
0.7699 - val_loss: 0.5549 - val_accuracy: 0.7283
```

[24]:
```
plot_results(results)
```

```
Maximum Loss : 1.6445

Minimum Loss : 0.4756

Loss difference : 1.1690
```





The training loss is decreasing, which is a positive sign, indicating that the model is learning from the training data. The validation loss is also decreasing, which is good. However, starting from around epoch 20, there seems to be an increase in the validation loss. This could be an indication of overfitting, and we will use a callback like earlystopping to tackle this problem, the training accuracy is increasing and has reached a high value and the validation accuracy shows some fluctuations but generally follows the training accuracy. By using early stopping it will also prevent the decreasing but fluctuating part. We can also use data augmentation technique to artificially increase the size of our training dataset and improve generalization.

Another important thing which may impact the result is the class imbalance, at the beginning of the data preprocessing we saw the distribution of classes was 72% for 0 label and 28% fir 1 label. Now that we are sure about the architecture of our model and some of basic hyper parameters we can calculate the weght of classes and apply that as well to see it will affect the performance or not.

[25]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.utils.class_weight import compute_class_weight
from tensorflow.keras.callbacks import EarlyStopping
```

[26]:
```
# Early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5,␣
 ↪restore_best_weights=True)

# Compute class weights and convert to dictionary format
class_labels = np.unique(train_labels)
class_weights = compute_class_weight(class_weight = "balanced",
                                     classes = np.unique(class_labels),
                                     y = train_labels)
class_weight_dict = dict(zip(class_labels, class_weights))
```

```python
# Data augmentation using ImageDataGenerator
# It works best when we are loading data from a local directory or CSV
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# generate augmented batches
train_datagen = datagen.flow(train_images, train_labels, batch_size=30)

results = model.fit(
    train_datagen,
    steps_per_epoch=len(train_images) // 30,
    epochs=40,
    validation_data=(valid_images, valid_labels),
    callbacks=[early_stopping],
    class_weight=class_weight_dict
)
```

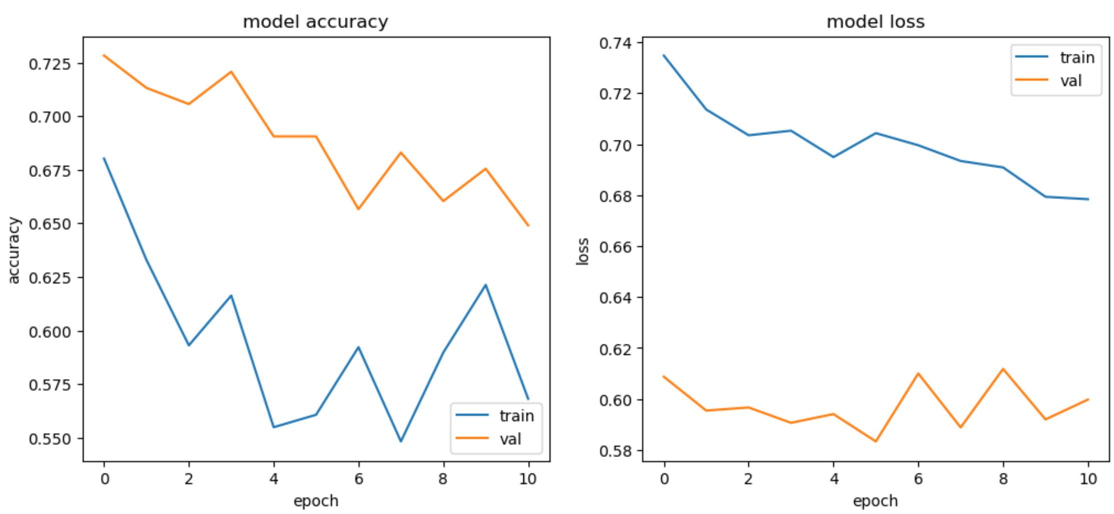
```
Epoch 1/40
41/41 [==============================] - 9s 196ms/step - loss: 0.7347 -
accuracy: 0.6802 - val_loss: 0.6087 - val_accuracy: 0.7283
Epoch 2/40
41/41 [==============================] - 8s 198ms/step - loss: 0.7136 -
accuracy: 0.6329 - val_loss: 0.5954 - val_accuracy: 0.7132
Epoch 3/40
41/41 [==============================] - 8s 185ms/step - loss: 0.7035 -
accuracy: 0.5930 - val_loss: 0.5967 - val_accuracy: 0.7057
Epoch 4/40
41/41 [==============================] - 7s 177ms/step - loss: 0.7053 -
accuracy: 0.6163 - val_loss: 0.5906 - val_accuracy: 0.7208
Epoch 5/40
41/41 [==============================] - 7s 173ms/step - loss: 0.6949 -
accuracy: 0.5548 - val_loss: 0.5941 - val_accuracy: 0.6906
Epoch 6/40
41/41 [==============================] - 7s 175ms/step - loss: 0.7043 -
accuracy: 0.5606 - val_loss: 0.5833 - val_accuracy: 0.6906
Epoch 7/40
41/41 [==============================] - 7s 179ms/step - loss: 0.6995 -
accuracy: 0.5922 - val_loss: 0.6100 - val_accuracy: 0.6566
Epoch 8/40
41/41 [==============================] - 7s 173ms/step - loss: 0.6934 -
```

```
accuracy: 0.5482 - val_loss: 0.5888 - val_accuracy: 0.6830
Epoch 9/40
41/41 [==============================] - 7s 180ms/step - loss: 0.6908 -
accuracy: 0.5897 - val_loss: 0.6118 - val_accuracy: 0.6604
Epoch 10/40
41/41 [==============================] - 8s 204ms/step - loss: 0.6793 -
accuracy: 0.6213 - val_loss: 0.5920 - val_accuracy: 0.6755
Epoch 11/40
41/41 [==============================] - 9s 221ms/step - loss: 0.6784 -
accuracy: 0.5681 - val_loss: 0.5998 - val_accuracy: 0.6491
```

[27]:

```
plot_results(results)
```

```
Maximum Loss : 0.7347

Minimum Loss : 0.6784

Loss difference : 0.0564
```



It looks like the changes didn't have a desirable impact, so far the best model was the previous one. Maybe we can find the best hyper parameters with use of grid search technique or make small changes to the architecture, so i will implement the grid search or play around with the previous model. I Will just add one other convolutional layer and one other dense layer.

[28]:

```
# First create the empty model
model = Sequential()

# Very important to clear the session to avoid mixing up with and being␣
↪affetced by previous computations
tf.keras.backend.clear_session()
```

```
# First convolutional layers gets the input which is resolution and channels in␣
 ↪our case -> (128,128,3)
# Using MaxPooling => reduce the spatial dimensions of the input,while␣
 ↪retaining the most important information
model.add(Conv2D(32, 3, kernel_initializer='normal', activation='relu',␣
 ↪input_shape=train_images[0].shape))
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Conv2D(32, 3, kernel_initializer='normal', activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Conv2D(64, 3, kernel_initializer='normal', activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Conv2D(128, 3, kernel_initializer='normal', activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Flatten())

# Using drop which out is another regularization technique
# dropout rate = 0.45 ==> 45% of the input units will be randomly set to zero␣
 ↪(Drop)
# Adding the kernel regulizer,l2 and increasing dropout rate
model.add(Dense(64, activation='relu',kernel_regularizer=l2(0.01)))
model.add(Dropout(0.5))

model.add(Dense(32, activation='relu',kernel_regularizer=l2(0.01)))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))
```

[29]:

```
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 32)      896

 max_pooling2d (MaxPooling2  (None, 62, 62, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 60, 60, 32)        9248

 max_pooling2d_1 (MaxPoolin  (None, 29, 29, 32)        0
 g2D)
```

```
 conv2d_2 (Conv2D)           (None, 27, 27, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 13, 13, 64)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 11, 11, 128)       73856

 max_pooling2d_3 (MaxPoolin  (None, 5, 5, 128)         0
 g2D)

 flatten (Flatten)           (None, 3200)              0

 dense (Dense)               (None, 64)                204864

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 32)                2080

 dropout_1 (Dropout)         (None, 32)                0

 dense_2 (Dense)             (None, 1)                 33

=================================================================
Total params: 309473 (1.18 MB)
Trainable params: 309473 (1.18 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[30]:

```
model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.0001),␣
 ↪metrics=['accuracy'])
results = model.fit(train_images,
             train_labels,
             epochs=40,
             batch_size=30,
             validation_data=(valid_images, valid_labels))
```

```
Epoch 1/40
42/42 [==============================] - 6s 111ms/step - loss: 2.1663 -
accuracy: 0.7107 - val_loss: 2.0402 - val_accuracy: 0.6830
Epoch 2/40
42/42 [==============================] - 4s 101ms/step - loss: 1.9218 -
accuracy: 0.7188 - val_loss: 1.8146 - val_accuracy: 0.6830
Epoch 3/40
42/42 [==============================] - 4s 99ms/step - loss: 1.7135 - accuracy:
0.7253 - val_loss: 1.6359 - val_accuracy: 0.6830
Epoch 4/40
42/42 [==============================] - 4s 101ms/step - loss: 1.5579 -
```

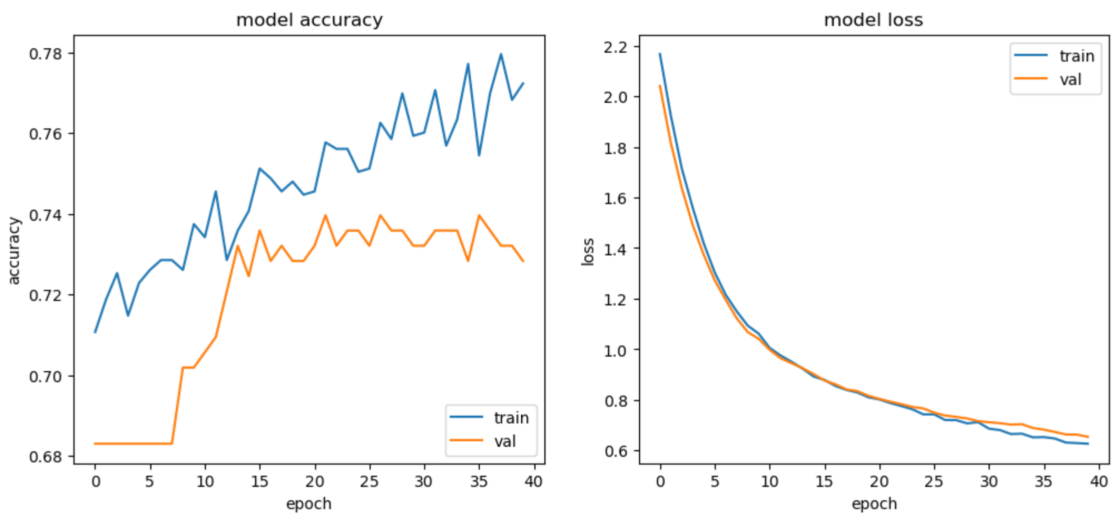
```
accuracy: 0.7147 - val_loss: 1.4899 - val_accuracy: 0.6830
Epoch 5/40
42/42 [==============================] - 4s 100ms/step - loss: 1.4179 -
accuracy: 0.7229 - val_loss: 1.3730 - val_accuracy: 0.6830
Epoch 6/40
42/42 [==============================] - 4s 100ms/step - loss: 1.3032 -
accuracy: 0.7261 - val_loss: 1.2724 - val_accuracy: 0.6830
Epoch 7/40
42/42 [==============================] - 4s 98ms/step - loss: 1.2152 - accuracy:
0.7285 - val_loss: 1.1952 - val_accuracy: 0.6830
Epoch 8/40
42/42 [==============================] - 4s 98ms/step - loss: 1.1501 - accuracy:
0.7285 - val_loss: 1.1229 - val_accuracy: 0.6830
Epoch 9/40
42/42 [==============================] - 4s 100ms/step - loss: 1.0938 -
accuracy: 0.7261 - val_loss: 1.0683 - val_accuracy: 0.7019
Epoch 10/40
42/42 [==============================] - 4s 100ms/step - loss: 1.0619 -
accuracy: 0.7374 - val_loss: 1.0416 - val_accuracy: 0.7019
Epoch 11/40
42/42 [==============================] - 4s 98ms/step - loss: 1.0056 - accuracy:
0.7342 - val_loss: 0.9983 - val_accuracy: 0.7057
Epoch 12/40
42/42 [==============================] - 4s 101ms/step - loss: 0.9755 -
accuracy: 0.7455 - val_loss: 0.9648 - val_accuracy: 0.7094
Epoch 13/40
42/42 [==============================] - 4s 99ms/step - loss: 0.9508 - accuracy:
0.7285 - val_loss: 0.9451 - val_accuracy: 0.7208
Epoch 14/40
42/42 [==============================] - 4s 101ms/step - loss: 0.9240 -
accuracy: 0.7358 - val_loss: 0.9255 - val_accuracy: 0.7321
Epoch 15/40
42/42 [==============================] - 4s 100ms/step - loss: 0.8927 -
accuracy: 0.7407 - val_loss: 0.9025 - val_accuracy: 0.7245
Epoch 16/40
42/42 [==============================] - 4s 100ms/step - loss: 0.8792 -
accuracy: 0.7512 - val_loss: 0.8768 - val_accuracy: 0.7358
Epoch 17/40
42/42 [==============================] - 4s 99ms/step - loss: 0.8551 - accuracy:
0.7488 - val_loss: 0.8615 - val_accuracy: 0.7283
Epoch 18/40
42/42 [==============================] - 4s 96ms/step - loss: 0.8395 - accuracy:
0.7455 - val_loss: 0.8417 - val_accuracy: 0.7321
Epoch 19/40
42/42 [==============================] - 4s 102ms/step - loss: 0.8295 -
accuracy: 0.7480 - val_loss: 0.8346 - val_accuracy: 0.7283
Epoch 20/40
42/42 [==============================] - 4s 102ms/step - loss: 0.8105 -
```

```
accuracy: 0.7447 - val_loss: 0.8163 - val_accuracy: 0.7283
Epoch 21/40
42/42 [==============================] - 4s 102ms/step - loss: 0.8021 -
accuracy: 0.7455 - val_loss: 0.8041 - val_accuracy: 0.7321
Epoch 22/40
42/42 [==============================] - 4s 98ms/step - loss: 0.7878 - accuracy:
0.7577 - val_loss: 0.7931 - val_accuracy: 0.7396
Epoch 23/40
42/42 [==============================] - 4s 98ms/step - loss: 0.7759 - accuracy:
0.7561 - val_loss: 0.7835 - val_accuracy: 0.7321
Epoch 24/40
42/42 [==============================] - 4s 101ms/step - loss: 0.7633 -
accuracy: 0.7561 - val_loss: 0.7722 - val_accuracy: 0.7358
Epoch 25/40
42/42 [==============================] - 4s 100ms/step - loss: 0.7431 -
accuracy: 0.7504 - val_loss: 0.7666 - val_accuracy: 0.7358
Epoch 26/40
42/42 [==============================] - 4s 98ms/step - loss: 0.7430 - accuracy:
0.7512 - val_loss: 0.7497 - val_accuracy: 0.7321
Epoch 27/40
42/42 [==============================] - 4s 103ms/step - loss: 0.7207 -
accuracy: 0.7626 - val_loss: 0.7383 - val_accuracy: 0.7396
Epoch 28/40
42/42 [==============================] - 4s 97ms/step - loss: 0.7203 - accuracy:
0.7585 - val_loss: 0.7330 - val_accuracy: 0.7358
Epoch 29/40
42/42 [==============================] - 4s 96ms/step - loss: 0.7075 - accuracy:
0.7699 - val_loss: 0.7269 - val_accuracy: 0.7358
Epoch 30/40
42/42 [==============================] - 4s 97ms/step - loss: 0.7121 - accuracy:
0.7593 - val_loss: 0.7163 - val_accuracy: 0.7321
Epoch 31/40
42/42 [==============================] - 4s 99ms/step - loss: 0.6864 - accuracy:
0.7601 - val_loss: 0.7118 - val_accuracy: 0.7321
Epoch 32/40
42/42 [==============================] - 4s 97ms/step - loss: 0.6808 - accuracy:
0.7707 - val_loss: 0.7082 - val_accuracy: 0.7358
Epoch 33/40
42/42 [==============================] - 4s 97ms/step - loss: 0.6652 - accuracy:
0.7569 - val_loss: 0.7025 - val_accuracy: 0.7358
Epoch 34/40
42/42 [==============================] - 4s 100ms/step - loss: 0.6666 -
accuracy: 0.7634 - val_loss: 0.7042 - val_accuracy: 0.7358
Epoch 35/40
42/42 [==============================] - 4s 101ms/step - loss: 0.6522 -
accuracy: 0.7771 - val_loss: 0.6891 - val_accuracy: 0.7283
Epoch 36/40
42/42 [==============================] - 4s 101ms/step - loss: 0.6532 -
```

```
accuracy: 0.7545 - val_loss: 0.6826 - val_accuracy: 0.7396
Epoch 37/40
42/42 [==============================] - 4s 100ms/step - loss: 0.6476 -
accuracy: 0.7699 - val_loss: 0.6736 - val_accuracy: 0.7358
Epoch 38/40
42/42 [==============================] - 4s 102ms/step - loss: 0.6316 -
accuracy: 0.7796 - val_loss: 0.6636 - val_accuracy: 0.7321
Epoch 39/40
42/42 [==============================] - 4s 101ms/step - loss: 0.6296 -
accuracy: 0.7682 - val_loss: 0.6631 - val_accuracy: 0.7321
Epoch 40/40
42/42 [==============================] - 4s 101ms/step - loss: 0.6271 -
accuracy: 0.7723 - val_loss: 0.6545 - val_accuracy: 0.7283
```

[31]:
```
plot_results(results)
```

```
Maximum Loss : 2.1663

Minimum Loss : 0.6271

Loss difference : 1.5392
```



I think finally everything seem to be better and the loss functions looks very good. In this case, there is clearly a healthy correlation between training loss and the validation loss. They both seem to reduce and stay at a constant value. This means that the model is well trained and is equally good on the training data as well as the validation data. However we can still try some strategies to increase the accuracy.

Learning Rate Schedule

[32]:

```
from tensorflow.keras.callbacks import LearningRateScheduler

def lr_schedule(epoch):
    return 0.001 * 0.9 ** epoch

lr_scheduler = LearningRateScheduler(lr_schedule)

results = model.fit(train_images,
             train_labels,
             epochs=40,
             batch_size=30,
             validation_data=(valid_images, valid_labels),
             callbacks=[lr_scheduler])
```

```
Epoch 1/40
42/42 [==============================] - 4s 102ms/step - loss: 0.6535 -
accuracy: 0.7455 - val_loss: 0.6632 - val_accuracy: 0.6981 - lr: 0.0010
Epoch 2/40
42/42 [==============================] - 4s 97ms/step - loss: 0.6231 - accuracy:
0.7512 - val_loss: 0.6316 - val_accuracy: 0.7321 - lr: 9.0000e-04
Epoch 3/40
42/42 [==============================] - 4s 101ms/step - loss: 0.5920 -
accuracy: 0.7520 - val_loss: 0.6049 - val_accuracy: 0.7509 - lr: 8.1000e-04
Epoch 4/40
42/42 [==============================] - 4s 98ms/step - loss: 0.5771 - accuracy:
0.7528 - val_loss: 0.6519 - val_accuracy: 0.7434 - lr: 7.2900e-04
Epoch 5/40
42/42 [==============================] - 4s 99ms/step - loss: 0.5722 - accuracy:
0.7585 - val_loss: 0.5924 - val_accuracy: 0.7472 - lr: 6.5610e-04
Epoch 6/40
42/42 [==============================] - 4s 103ms/step - loss: 0.5477 -
accuracy: 0.7618 - val_loss: 0.5737 - val_accuracy: 0.7358 - lr: 5.9049e-04
Epoch 7/40
42/42 [==============================] - 4s 99ms/step - loss: 0.5319 - accuracy:
0.7780 - val_loss: 0.5822 - val_accuracy: 0.7396 - lr: 5.3144e-04
Epoch 8/40
42/42 [==============================] - 4s 102ms/step - loss: 0.5261 -
accuracy: 0.7626 - val_loss: 0.5690 - val_accuracy: 0.7472 - lr: 4.7830e-04
Epoch 9/40
42/42 [==============================] - 4s 101ms/step - loss: 0.5152 -
accuracy: 0.7771 - val_loss: 0.5707 - val_accuracy: 0.7434 - lr: 4.3047e-04
Epoch 10/40
42/42 [==============================] - 4s 102ms/step - loss: 0.5207 -
accuracy: 0.7780 - val_loss: 0.5728 - val_accuracy: 0.7623 - lr: 3.8742e-04
Epoch 11/40
42/42 [==============================] - 4s 105ms/step - loss: 0.4953 -
accuracy: 0.7885 - val_loss: 0.5751 - val_accuracy: 0.7472 - lr: 3.4868e-04
Epoch 12/40
```

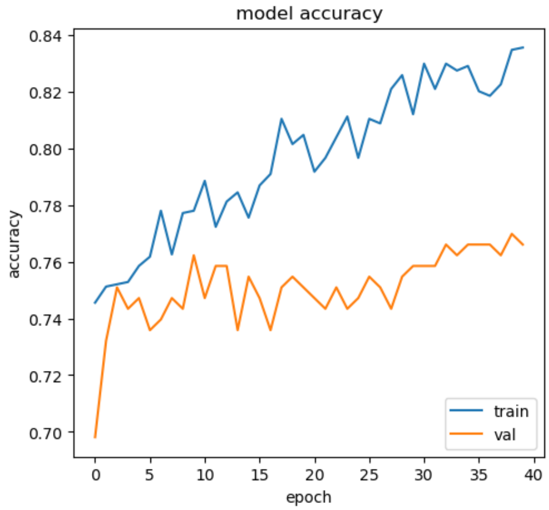
```
42/42 [==============================] - 4s 100ms/step - loss: 0.4764 -
accuracy: 0.7723 - val_loss: 0.5639 - val_accuracy: 0.7585 - lr: 3.1381e-04
Epoch 13/40
42/42 [==============================] - 4s 100ms/step - loss: 0.4887 -
accuracy: 0.7812 - val_loss: 0.5733 - val_accuracy: 0.7585 - lr: 2.8243e-04
Epoch 14/40
42/42 [==============================] - 4s 102ms/step - loss: 0.4814 -
accuracy: 0.7844 - val_loss: 0.5841 - val_accuracy: 0.7358 - lr: 2.5419e-04
Epoch 15/40
42/42 [==============================] - 4s 102ms/step - loss: 0.4738 -
accuracy: 0.7755 - val_loss: 0.5873 - val_accuracy: 0.7547 - lr: 2.2877e-04
Epoch 16/40
42/42 [==============================] - 4s 100ms/step - loss: 0.4773 -
accuracy: 0.7869 - val_loss: 0.5756 - val_accuracy: 0.7472 - lr: 2.0589e-04
Epoch 17/40
42/42 [==============================] - 4s 100ms/step - loss: 0.4550 -
accuracy: 0.7909 - val_loss: 0.5969 - val_accuracy: 0.7358 - lr: 1.8530e-04
Epoch 18/40
42/42 [==============================] - 4s 98ms/step - loss: 0.4501 - accuracy:
0.8104 - val_loss: 0.6059 - val_accuracy: 0.7509 - lr: 1.6677e-04
Epoch 19/40
42/42 [==============================] - 4s 101ms/step - loss: 0.4559 -
accuracy: 0.8015 - val_loss: 0.5892 - val_accuracy: 0.7547 - lr: 1.5009e-04
Epoch 20/40
42/42 [==============================] - 4s 99ms/step - loss: 0.4447 - accuracy:
0.8047 - val_loss: 0.6063 - val_accuracy: 0.7509 - lr: 1.3509e-04
Epoch 21/40
42/42 [==============================] - 4s 98ms/step - loss: 0.4395 - accuracy:
0.7917 - val_loss: 0.6163 - val_accuracy: 0.7472 - lr: 1.2158e-04
Epoch 22/40
42/42 [==============================] - 4s 99ms/step - loss: 0.4405 - accuracy:
0.7966 - val_loss: 0.6136 - val_accuracy: 0.7434 - lr: 1.0942e-04
Epoch 23/40
42/42 [==============================] - 4s 98ms/step - loss: 0.4350 - accuracy:
0.8039 - val_loss: 0.6251 - val_accuracy: 0.7509 - lr: 9.8477e-05
Epoch 24/40
42/42 [==============================] - 4s 99ms/step - loss: 0.4253 - accuracy:
0.8112 - val_loss: 0.6289 - val_accuracy: 0.7434 - lr: 8.8629e-05
Epoch 25/40
42/42 [==============================] - 4s 102ms/step - loss: 0.4281 -
accuracy: 0.7966 - val_loss: 0.6126 - val_accuracy: 0.7472 - lr: 7.9766e-05
Epoch 26/40
42/42 [==============================] - 4s 101ms/step - loss: 0.4167 -
accuracy: 0.8104 - val_loss: 0.6642 - val_accuracy: 0.7547 - lr: 7.1790e-05
Epoch 27/40
42/42 [==============================] - 4s 100ms/step - loss: 0.4293 -
accuracy: 0.8088 - val_loss: 0.6226 - val_accuracy: 0.7509 - lr: 6.4611e-05
Epoch 28/40
```

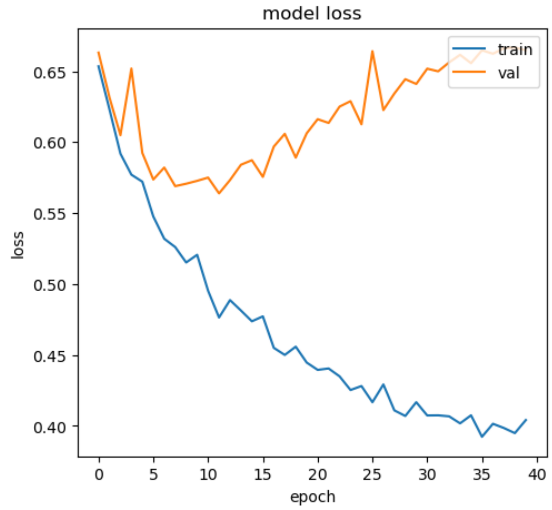
```
42/42 [==============================] - 4s 97ms/step - loss: 0.4110 - accuracy:
0.8209 - val_loss: 0.6343 - val_accuracy: 0.7434 - lr: 5.8150e-05
Epoch 29/40
42/42 [==============================] - 4s 96ms/step - loss: 0.4070 - accuracy:
0.8258 - val_loss: 0.6445 - val_accuracy: 0.7547 - lr: 5.2335e-05
Epoch 30/40
42/42 [==============================] - 4s 98ms/step - loss: 0.4168 - accuracy:
0.8120 - val_loss: 0.6411 - val_accuracy: 0.7585 - lr: 4.7101e-05
Epoch 31/40
42/42 [==============================] - 4s 98ms/step - loss: 0.4074 - accuracy:
0.8298 - val_loss: 0.6519 - val_accuracy: 0.7585 - lr: 4.2391e-05
Epoch 32/40
42/42 [==============================] - 4s 100ms/step - loss: 0.4075 -
accuracy: 0.8209 - val_loss: 0.6500 - val_accuracy: 0.7585 - lr: 3.8152e-05
Epoch 33/40
42/42 [==============================] - 4s 100ms/step - loss: 0.4068 -
accuracy: 0.8298 - val_loss: 0.6563 - val_accuracy: 0.7660 - lr: 3.4337e-05
Epoch 34/40
42/42 [==============================] - 4s 102ms/step - loss: 0.4018 -
accuracy: 0.8274 - val_loss: 0.6616 - val_accuracy: 0.7623 - lr: 3.0903e-05
Epoch 35/40
42/42 [==============================] - 4s 101ms/step - loss: 0.4075 -
accuracy: 0.8290 - val_loss: 0.6557 - val_accuracy: 0.7660 - lr: 2.7813e-05
Epoch 36/40
42/42 [==============================] - 4s 97ms/step - loss: 0.3924 - accuracy:
0.8201 - val_loss: 0.6648 - val_accuracy: 0.7660 - lr: 2.5032e-05
Epoch 37/40
42/42 [==============================] - 4s 101ms/step - loss: 0.4015 -
accuracy: 0.8185 - val_loss: 0.6624 - val_accuracy: 0.7660 - lr: 2.2528e-05
Epoch 38/40
42/42 [==============================] - 4s 100ms/step - loss: 0.3986 -
accuracy: 0.8225 - val_loss: 0.6662 - val_accuracy: 0.7623 - lr: 2.0276e-05
Epoch 39/40
42/42 [==============================] - 4s 100ms/step - loss: 0.3950 -
accuracy: 0.8347 - val_loss: 0.6668 - val_accuracy: 0.7698 - lr: 1.8248e-05
Epoch 40/40
42/42 [==============================] - 4s 99ms/step - loss: 0.4042 - accuracy:
0.8355 - val_loss: 0.6648 - val_accuracy: 0.7660 - lr: 1.6423e-05
```

[33]:

```
plot_results(results)
```

```
Maximum Loss : 0.6535

Minimum Loss : 0.3924

Loss difference : 0.2611
```





Not a good solution at all for the reasons that i told before.

Batch Normalization

[34]:
```
# First create the empty model
model = Sequential()

# Very important to clear the session to avoid mixing up with and being␣
↪affetced by previous computations
tf.keras.backend.clear_session()

# First convolutional layers gets the input which is resolution and channels in␣
↪our case -> (128,128,3)
# Using MaxPooling => reduce the spatial dimensions of the input,while␣
↪retaining the most important information
model.add(Conv2D(32, 3, kernel_initializer='normal', activation='relu',␣
↪input_shape=train_images[0].shape))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Conv2D(32, 3, kernel_initializer='normal', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Conv2D(64, 3, kernel_initializer='normal', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Conv2D(128, 3, kernel_initializer='normal', activation='relu'))
model.add(BatchNormalization())
```

```
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Flatten())

# Using drop which out is another regularization technique
# dropout rate = 0.45 ==> 45% of the input units will be randomly set to zero␣
↪(Drop)
# Adding the kernel regulizer,l2 and increasing dropout rate
model.add(Dense(64, activation='relu',kernel_regularizer=l2(0.01)))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(32, activation='relu',kernel_regularizer=l2(0.01)))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))
```

[35]:
```
model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.0001),␣
↪metrics=['accuracy'])
results = model.fit(train_images,
             train_labels,
             epochs=40,
             batch_size=30,
             validation_data=(valid_images, valid_labels))
```

```
Epoch 1/40
42/42 [==============================] - 8s 121ms/step - loss: 2.5834 -
accuracy: 0.5235 - val_loss: 2.4330 - val_accuracy: 0.3170
Epoch 2/40
42/42 [==============================] - 5s 124ms/step - loss: 2.4953 -
accuracy: 0.5689 - val_loss: 2.3904 - val_accuracy: 0.3170
Epoch 3/40
42/42 [==============================] - 5s 124ms/step - loss: 2.4653 -
accuracy: 0.5932 - val_loss: 2.3735 - val_accuracy: 0.3170
Epoch 4/40
42/42 [==============================] - 5s 122ms/step - loss: 2.4091 -
accuracy: 0.6094 - val_loss: 2.3310 - val_accuracy: 0.4491
Epoch 5/40
42/42 [==============================] - 5s 124ms/step - loss: 2.3842 -
accuracy: 0.6248 - val_loss: 2.3060 - val_accuracy: 0.5774
Epoch 6/40
42/42 [==============================] - 5s 124ms/step - loss: 2.3590 -
accuracy: 0.5981 - val_loss: 2.2845 - val_accuracy: 0.6000
Epoch 7/40
42/42 [==============================] - 5s 124ms/step - loss: 2.3096 -
accuracy: 0.6329 - val_loss: 2.2634 - val_accuracy: 0.6264
Epoch 8/40
```

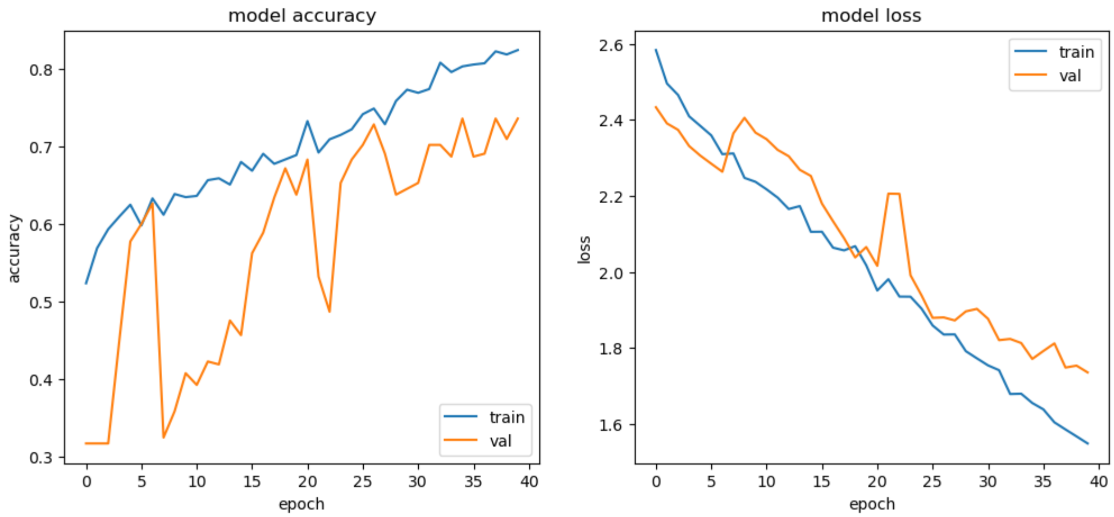
```
42/42 [==============================] - 5s 124ms/step - loss: 2.3115 -
accuracy: 0.6118 - val_loss: 2.3644 - val_accuracy: 0.3245
Epoch 9/40
42/42 [==============================] - 5s 124ms/step - loss: 2.2472 -
accuracy: 0.6386 - val_loss: 2.4052 - val_accuracy: 0.3585
Epoch 10/40
42/42 [==============================] - 5s 125ms/step - loss: 2.2368 -
accuracy: 0.6345 - val_loss: 2.3661 - val_accuracy: 0.4075
Epoch 11/40
42/42 [==============================] - 5s 127ms/step - loss: 2.2170 -
accuracy: 0.6361 - val_loss: 2.3489 - val_accuracy: 0.3925
Epoch 12/40
42/42 [==============================] - 5s 127ms/step - loss: 2.1952 -
accuracy: 0.6564 - val_loss: 2.3208 - val_accuracy: 0.4226
Epoch 13/40
42/42 [==============================] - 5s 130ms/step - loss: 2.1652 -
accuracy: 0.6588 - val_loss: 2.3041 - val_accuracy: 0.4189
Epoch 14/40
42/42 [==============================] - 5s 130ms/step - loss: 2.1731 -
accuracy: 0.6507 - val_loss: 2.2684 - val_accuracy: 0.4755
Epoch 15/40
42/42 [==============================] - 5s 126ms/step - loss: 2.1053 -
accuracy: 0.6799 - val_loss: 2.2522 - val_accuracy: 0.4566
Epoch 16/40
42/42 [==============================] - 5s 127ms/step - loss: 2.1055 -
accuracy: 0.6686 - val_loss: 2.1796 - val_accuracy: 0.5623
Epoch 17/40
42/42 [==============================] - 5s 128ms/step - loss: 2.0637 -
accuracy: 0.6904 - val_loss: 2.1332 - val_accuracy: 0.5887
Epoch 18/40
42/42 [==============================] - 5s 125ms/step - loss: 2.0565 -
accuracy: 0.6775 - val_loss: 2.0884 - val_accuracy: 0.6340
Epoch 19/40
42/42 [==============================] - 5s 125ms/step - loss: 2.0676 -
accuracy: 0.6831 - val_loss: 2.0379 - val_accuracy: 0.6717
Epoch 20/40
42/42 [==============================] - 5s 131ms/step - loss: 2.0175 -
accuracy: 0.6888 - val_loss: 2.0653 - val_accuracy: 0.6377
Epoch 21/40
42/42 [==============================] - 5s 126ms/step - loss: 1.9510 -
accuracy: 0.7326 - val_loss: 2.0159 - val_accuracy: 0.6830
Epoch 22/40
42/42 [==============================] - 5s 128ms/step - loss: 1.9808 -
accuracy: 0.6921 - val_loss: 2.2057 - val_accuracy: 0.5321
Epoch 23/40
42/42 [==============================] - 5s 125ms/step - loss: 1.9350 -
accuracy: 0.7091 - val_loss: 2.2052 - val_accuracy: 0.4868
Epoch 24/40
```

```
42/42 [==============================] - 5s 129ms/step - loss: 1.9347 -
accuracy: 0.7147 - val_loss: 1.9911 - val_accuracy: 0.6528
Epoch 25/40
42/42 [==============================] - 5s 131ms/step - loss: 1.9032 -
accuracy: 0.7220 - val_loss: 1.9380 - val_accuracy: 0.6830
Epoch 26/40
42/42 [==============================] - 5s 126ms/step - loss: 1.8590 -
accuracy: 0.7415 - val_loss: 1.8790 - val_accuracy: 0.7019
Epoch 27/40
42/42 [==============================] - 5s 128ms/step - loss: 1.8352 -
accuracy: 0.7488 - val_loss: 1.8802 - val_accuracy: 0.7283
Epoch 28/40
42/42 [==============================] - 5s 124ms/step - loss: 1.8356 -
accuracy: 0.7285 - val_loss: 1.8723 - val_accuracy: 0.6906
Epoch 29/40
42/42 [==============================] - 5s 128ms/step - loss: 1.7914 -
accuracy: 0.7585 - val_loss: 1.8961 - val_accuracy: 0.6377
Epoch 30/40
42/42 [==============================] - 5s 127ms/step - loss: 1.7725 -
accuracy: 0.7731 - val_loss: 1.9027 - val_accuracy: 0.6453
Epoch 31/40
42/42 [==============================] - 5s 126ms/step - loss: 1.7541 -
accuracy: 0.7690 - val_loss: 1.8766 - val_accuracy: 0.6528
Epoch 32/40
42/42 [==============================] - 5s 129ms/step - loss: 1.7414 -
accuracy: 0.7739 - val_loss: 1.8204 - val_accuracy: 0.7019
Epoch 33/40
42/42 [==============================] - 5s 126ms/step - loss: 1.6787 -
accuracy: 0.8079 - val_loss: 1.8236 - val_accuracy: 0.7019
Epoch 34/40
42/42 [==============================] - 5s 128ms/step - loss: 1.6795 -
accuracy: 0.7958 - val_loss: 1.8128 - val_accuracy: 0.6868
Epoch 35/40
42/42 [==============================] - 5s 126ms/step - loss: 1.6551 -
accuracy: 0.8031 - val_loss: 1.7710 - val_accuracy: 0.7358
Epoch 36/40
42/42 [==============================] - 5s 129ms/step - loss: 1.6384 -
accuracy: 0.8055 - val_loss: 1.7918 - val_accuracy: 0.6868
Epoch 37/40
42/42 [==============================] - 5s 128ms/step - loss: 1.6042 -
accuracy: 0.8071 - val_loss: 1.8119 - val_accuracy: 0.6906
Epoch 38/40
42/42 [==============================] - 5s 125ms/step - loss: 1.5855 -
accuracy: 0.8225 - val_loss: 1.7483 - val_accuracy: 0.7358
Epoch 39/40
42/42 [==============================] - 5s 127ms/step - loss: 1.5672 -
accuracy: 0.8185 - val_loss: 1.7534 - val_accuracy: 0.7094
Epoch 40/40
```

```
42/42 [==============================] - 6s 140ms/step - loss: 1.5487 -
accuracy: 0.8241 - val_loss: 1.7355 - val_accuracy: 0.7358
```

[36]:
```
plot_results(results)
```

```
Maximum Loss : 2.5834

Minimum Loss : 1.5487

Loss difference : 1.0346
```



Same problems, doesnt look a good solution at all i will move further with another technique

Weight Initialization

[37]:
```
from tensorflow.keras.initializers import he_normal

# First create the empty model
model = Sequential()

# Very important to clear the session to avoid mixing up with and being
 ↪affetced by previous computations
tf.keras.backend.clear_session()

# First convolutional layers gets the input which is resolution and channels in
 ↪our case -> (128,128,3)
# Using MaxPooling => reduce the spatial dimensions of the input,while
 ↪retaining the most important information
```

```
model.add(Conv2D(32, 3, kernel_initializer=he_normal(), activation='relu',␣
↪input_shape=train_images[0].shape))
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Conv2D(32, 3, kernel_initializer=he_normal(), activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Conv2D(64, 3, kernel_initializer=he_normal(), activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Conv2D(128, 3, kernel_initializer=he_normal(), activation='relu'))
model.add(MaxPooling2D(pool_size=(3, 3), strides=2))

model.add(Flatten())

# Using drop which out is another regularization technique
# dropout rate = 0.45 ==> 45% of the input units will be randomly set to zero␣
↪(Drop)
# Adding the kernel regulizer,l2 and increasing dropout rate
model.add(Dense(64, activation='relu',kernel_regularizer=l2(0.01)))
model.add(Dropout(0.5))

model.add(Dense(32, activation='relu',kernel_regularizer=l2(0.01)))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))
```

[38]:
```
model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.0001),␣
↪metrics=['accuracy'])
results = model.fit(train_images,
             train_labels,
             epochs=40,
             batch_size=30,
             validation_data=(valid_images, valid_labels))
```

```
Epoch 1/40
42/42 [==============================] - 7s 113ms/step - loss: 2.2098 -
accuracy: 0.6694 - val_loss: 2.0307 - val_accuracy: 0.6830
Epoch 2/40
42/42 [==============================] - 4s 102ms/step - loss: 1.9331 -
accuracy: 0.7091 - val_loss: 1.8195 - val_accuracy: 0.6830
Epoch 3/40
42/42 [==============================] - 4s 99ms/step - loss: 1.7299 - accuracy:
0.7172 - val_loss: 1.6564 - val_accuracy: 0.6830
Epoch 4/40
42/42 [==============================] - 4s 100ms/step - loss: 1.6127 -
accuracy: 0.7123 - val_loss: 1.5424 - val_accuracy: 0.7283
```

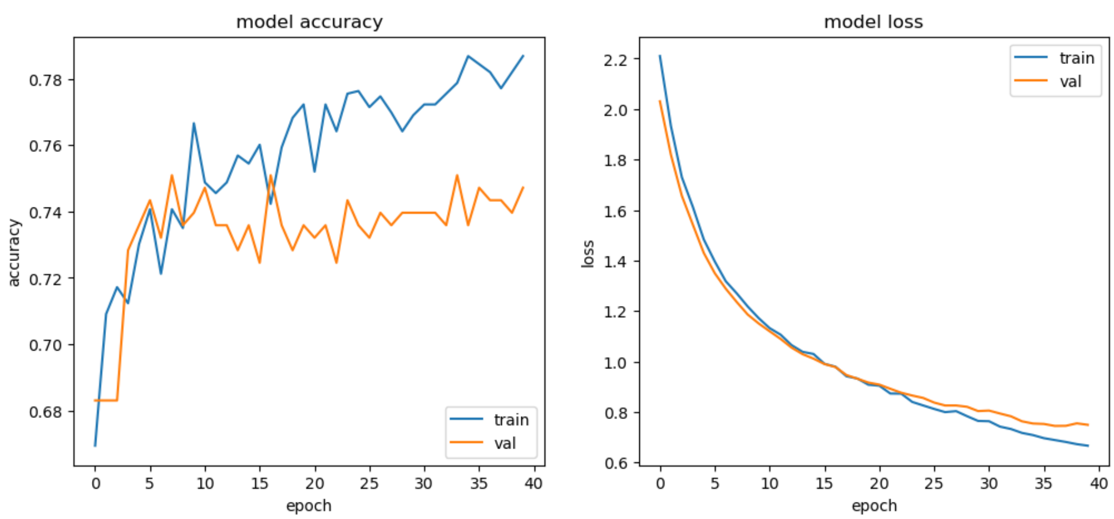
```
Epoch 5/40
42/42 [==============================] - 4s 101ms/step - loss: 1.4833 -
accuracy: 0.7301 - val_loss: 1.4299 - val_accuracy: 0.7358
Epoch 6/40
42/42 [==============================] - 4s 97ms/step - loss: 1.3964 - accuracy:
0.7407 - val_loss: 1.3499 - val_accuracy: 0.7434
Epoch 7/40
42/42 [==============================] - 4s 98ms/step - loss: 1.3182 - accuracy:
0.7212 - val_loss: 1.2886 - val_accuracy: 0.7321
Epoch 8/40
42/42 [==============================] - 4s 99ms/step - loss: 1.2701 - accuracy:
0.7407 - val_loss: 1.2358 - val_accuracy: 0.7509
Epoch 9/40
42/42 [==============================] - 4s 98ms/step - loss: 1.2181 - accuracy:
0.7350 - val_loss: 1.1862 - val_accuracy: 0.7358
Epoch 10/40
42/42 [==============================] - 4s 98ms/step - loss: 1.1720 - accuracy:
0.7666 - val_loss: 1.1508 - val_accuracy: 0.7396
Epoch 11/40
42/42 [==============================] - 4s 98ms/step - loss: 1.1323 - accuracy:
0.7488 - val_loss: 1.1198 - val_accuracy: 0.7472
Epoch 12/40
42/42 [==============================] - 4s 98ms/step - loss: 1.1063 - accuracy:
0.7455 - val_loss: 1.0896 - val_accuracy: 0.7358
Epoch 13/40
42/42 [==============================] - 4s 99ms/step - loss: 1.0649 - accuracy:
0.7488 - val_loss: 1.0550 - val_accuracy: 0.7358
Epoch 14/40
42/42 [==============================] - 4s 96ms/step - loss: 1.0385 - accuracy:
0.7569 - val_loss: 1.0292 - val_accuracy: 0.7283
Epoch 15/40
42/42 [==============================] - 4s 95ms/step - loss: 1.0299 - accuracy:
0.7545 - val_loss: 1.0114 - val_accuracy: 0.7358
Epoch 16/40
42/42 [==============================] - 4s 101ms/step - loss: 0.9909 -
accuracy: 0.7601 - val_loss: 0.9900 - val_accuracy: 0.7245
Epoch 17/40
42/42 [==============================] - 4s 100ms/step - loss: 0.9795 -
accuracy: 0.7423 - val_loss: 0.9781 - val_accuracy: 0.7509
Epoch 18/40
42/42 [==============================] - 4s 96ms/step - loss: 0.9417 - accuracy:
0.7593 - val_loss: 0.9464 - val_accuracy: 0.7358
Epoch 19/40
42/42 [==============================] - 4s 96ms/step - loss: 0.9327 - accuracy:
0.7682 - val_loss: 0.9320 - val_accuracy: 0.7283
Epoch 20/40
42/42 [==============================] - 4s 98ms/step - loss: 0.9082 - accuracy:
0.7723 - val_loss: 0.9167 - val_accuracy: 0.7358
```

```
Epoch 21/40
42/42 [==============================] - 4s 96ms/step - loss: 0.9043 - accuracy:
0.7520 - val_loss: 0.9085 - val_accuracy: 0.7321
Epoch 22/40
42/42 [==============================] - 4s 96ms/step - loss: 0.8733 - accuracy:
0.7723 - val_loss: 0.8921 - val_accuracy: 0.7358
Epoch 23/40
42/42 [==============================] - 4s 99ms/step - loss: 0.8728 - accuracy:
0.7642 - val_loss: 0.8764 - val_accuracy: 0.7245
Epoch 24/40
42/42 [==============================] - 4s 96ms/step - loss: 0.8400 - accuracy:
0.7755 - val_loss: 0.8653 - val_accuracy: 0.7434
Epoch 25/40
42/42 [==============================] - 4s 97ms/step - loss: 0.8260 - accuracy:
0.7763 - val_loss: 0.8557 - val_accuracy: 0.7358
Epoch 26/40
42/42 [==============================] - 4s 98ms/step - loss: 0.8122 - accuracy:
0.7715 - val_loss: 0.8374 - val_accuracy: 0.7321
Epoch 27/40
42/42 [==============================] - 4s 98ms/step - loss: 0.7995 - accuracy:
0.7747 - val_loss: 0.8258 - val_accuracy: 0.7396
Epoch 28/40
42/42 [==============================] - 4s 96ms/step - loss: 0.8035 - accuracy:
0.7699 - val_loss: 0.8259 - val_accuracy: 0.7358
Epoch 29/40
42/42 [==============================] - 4s 95ms/step - loss: 0.7835 - accuracy:
0.7642 - val_loss: 0.8207 - val_accuracy: 0.7396
Epoch 30/40
42/42 [==============================] - 4s 98ms/step - loss: 0.7647 - accuracy:
0.7690 - val_loss: 0.8038 - val_accuracy: 0.7396
Epoch 31/40
42/42 [==============================] - 4s 98ms/step - loss: 0.7635 - accuracy:
0.7723 - val_loss: 0.8057 - val_accuracy: 0.7396
Epoch 32/40
42/42 [==============================] - 4s 101ms/step - loss: 0.7421 -
accuracy: 0.7723 - val_loss: 0.7940 - val_accuracy: 0.7396
Epoch 33/40
42/42 [==============================] - 4s 98ms/step - loss: 0.7327 - accuracy:
0.7755 - val_loss: 0.7830 - val_accuracy: 0.7358
Epoch 34/40
42/42 [==============================] - 4s 97ms/step - loss: 0.7173 - accuracy:
0.7788 - val_loss: 0.7632 - val_accuracy: 0.7509
Epoch 35/40
42/42 [==============================] - 4s 101ms/step - loss: 0.7087 -
accuracy: 0.7869 - val_loss: 0.7545 - val_accuracy: 0.7358
Epoch 36/40
42/42 [==============================] - 4s 100ms/step - loss: 0.6965 -
accuracy: 0.7844 - val_loss: 0.7528 - val_accuracy: 0.7472
```

```
Epoch 37/40
42/42 [==============================] - 4s 101ms/step - loss: 0.6890 -
accuracy: 0.7820 - val_loss: 0.7449 - val_accuracy: 0.7434
Epoch 38/40
42/42 [==============================] - 4s 99ms/step - loss: 0.6814 - accuracy:
0.7771 - val_loss: 0.7453 - val_accuracy: 0.7434
Epoch 39/40
42/42 [==============================] - 4s 98ms/step - loss: 0.6727 - accuracy:
0.7820 - val_loss: 0.7552 - val_accuracy: 0.7396
Epoch 40/40
42/42 [==============================] - 4s 101ms/step - loss: 0.6666 -
accuracy: 0.7869 - val_loss: 0.7493 - val_accuracy: 0.7472
```

[39]:
```
plot_results(results)
```

```
Maximum Loss : 2.2098

Minimum Loss : 0.6666

Loss difference : 1.5432
```



Seems like it's getting better the validation accuracy is getting closer to train accuracy as well as the loss which is a great sign. I think this is the moment that i will give up cause approximately i did whatever i knew but i know there are lotfs of other things that we can do to modify our model. At the end i will save the model and do the prediction on the test data to check wether it has the same accuracy of our model which is about 75%. The closer this two number, the better because a huge difference may indicate the overfitting of our model

[40]:
```
from tensorflow.keras.models import load_model
from sklearn.metrics import accuracy_score
```

```
loaded_model = load_model("final_version")

predictions = loaded_model.predict(test_images)

# Convert the predictions to binary values (0 or 1) based on a threshold (e.g.,␣
↪0.5)
binary_predictions = (predictions > 0.5).astype(int)

accuracy = accuracy_score(test_labels, binary_predictions)

print(f"Accuracy: {accuracy}")
```

```
WARNING:tensorflow:From C:\Users\malir\anaconda3\lib\site-
packages\keras\src\saving\legacy\saved_model\load.py:107: The name
tf.gfile.Exists is deprecated. Please use tf.io.gfile.exists instead.

9/9 [==============================] - 0s 24ms/step
Accuracy: 0.7358490566037735
```

[ ]:

```

```