1.Importing required Libraries

In [1]:
```
pip install livelossplot
```

...

In [35]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [22]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [24]:
```
from sklearn import metrics
```

In [3]:
```
from livelossplot.inputs.keras import PlotLossesCallback
```

In [4]:
```
import tensorflow as tf
```

In [5]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

In [6]:
```
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Input, Dropout
from tensorflow.keras.layers import BatchNormalization, Activation, MaxPooling2D


from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.callbacks import ModelCheckpoint
```

2.Load and Pre-process the dataset

▶Loading the dataset using the pandas read_csv function

In [26]:
```
train_images = pd.read_csv("train_images.csv", header=None)
train_labels = pd.read_csv("train_labels.csv", header=None)
val_images = pd.read_csv("val_images.csv", header=None)
val_labels = pd.read_csv("val_labels.csv", header=None)
```

In [8]:
```
train_images.head()
```

Out[8]:

	0	1	2	3	4	5	6	7	8	9	...	8182	8183	8184	8185
0	0.631373	0.623529	0.713726	0.705882	0.658824	0.666667	0.654902	0.635294	0.647059	0.705882	...	0.682353	0.611765	0.650980	0.658824
1	0.725490	0.752941	0.749020	0.701961	0.690196	0.721569	0.709804	0.745098	0.654902	0.721569	...	0.721569	0.698039	0.721569	0.686275
2	0.717647	0.701961	0.713726	0.733333	0.705882	0.717647	0.725490	0.682353	0.717647	0.674510	...	0.709804	0.694118	0.705882	0.682353
3	0.705882	0.674510	0.654902	0.678431	0.666667	0.662745	0.678431	0.662745	0.686275	0.686275	...	0.639216	0.662745	0.631373	0.643137
4	0.647059	0.729412	0.701961	0.674510	0.611765	0.698039	0.713726	0.662745	0.701961	0.674510	...	0.639216	0.670588	0.705882	0.674510

5 rows × 8192 columns

In [9]:
```
train_labels.head()
```

Out[9]:

	0	1	2	3
0	1.0	0.0	0.0	0.0
1	1.0	0.0	0.0	0.0
2	1.0	0.0	0.0	0.0
3	1.0	0.0	0.0	0.0
4	1.0	0.0	0.0	0.0

▶Checking the shape of the training and validation data

In [10]:
```
print(train_images.shape, train_labels.shape)
```

```
(3200, 8192) (3200, 4)
```

In [11]:
```
print(val_images.shape, val_labels.shape)
```

```
(800, 8192) (800, 4)
```

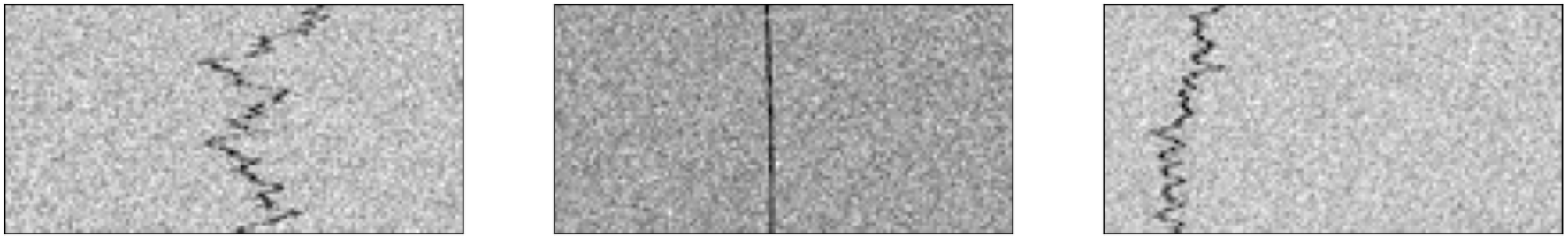
▶Reshaping the training and validation images

In [12]:
```
x_train = train_images.values.reshape(3200, 64, 128, 1)
x_val = val_images.values.reshape(800, 64, 128, 1)

y_train = train_labels.values
y_val = val_labels.values
```

3.Visualize the dataset

In [13]:
```
plt.figure(0, figsize=(15, 15))
for i in range(1,4):
  plt.subplot(1, 3, i)
  img = np.squeeze(x_train[np.random.randint(0, x_train.shape[0])])
  plt.xticks([])
  plt.yticks([])
  plt.imshow(img, cmap="gray")
```



4.Create Training and Validation Data Generators using Keras ImageDataGenerator function

In [14]:
```
datagen_train = ImageDataGenerator(horizontal_flip=True,
                                   rotation_range=0.2,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2)
datagen_train.fit(x_train)

datagen_val = ImageDataGenerator(horizontal_flip=True,
                                   rotation_range=0.2,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2)
datagen_val.fit(x_val)
```

5.Design a Convolutional Neural Network (CNN) Model

In [15]:

```
# INTIALIZING THE CNN
model = Sequential()

# FIRST CONVOLUTION
model.add(Conv2D(32, (5,5), padding='same', input_shape=(64, 128, 1)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))

# 2ND CONVOLUTION
model.add(Conv2D(64, (5,5), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))

# FLATTEN
model.add(Flatten())

# FULLY CONNECTED LAYER
model.add(Dense(1024))
model.add(Activation('relu'))
model.add(Dropout(0.5))

# OUTPUT LAYER
model.add(Dense(4, activation='softmax'))
```

6.Compile the Model using Adam optimizer, categorical_crossentropy loss function, and accuracy matrix

In [16]:
```
initial_learning_rate = 0.005

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate = initial_learning_rate,
    decay_steps = 5,
    decay_rate = 0.96,
    staircase=True
)

optimizer = Adam(learning_rate = lr_schedule)
```

In [17]:
```
model.compile(optimizer=optimizer,
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

7.Print the Model summary

In [18]:

```
1 model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 64, 128, 32)       832

 batch_normalization (Batch  (None, 64, 128, 32)       128
 Normalization)

 activation (Activation)     (None, 64, 128, 32)       0

 max_pooling2d (MaxPooling2  (None, 32, 64, 32)        0
 D)

 dropout (Dropout)           (None, 32, 64, 32)        0

 conv2d_1 (Conv2D)           (None, 32, 64, 64)        51264

 batch_normalization_1 (Bat  (None, 32, 64, 64)        256
 chNormalization)

 activation_1 (Activation)   (None, 32, 64, 64)        0

 max_pooling2d_1 (MaxPoolin  (None, 16, 32, 64)        0
 g2D)

 dropout_1 (Dropout)         (None, 16, 32, 64)        0

 flatten (Flatten)           (None, 32768)             0

 dense (Dense)               (None, 1024)              33555456

 activation_2 (Activation)   (None, 1024)              0

 dropout_2 (Dropout)         (None, 1024)              0

 dense_1 (Dense)             (None, 4)                 4100

=================================================================
Total params: 33612036 (128.22 MB)
Trainable params: 33611844 (128.22 MB)
```

```
Non-trainable params: 192 (768.00 Byte)
_________________________________________________________________
```

8.Training the Model with batch_size = 32 & epochs = 12

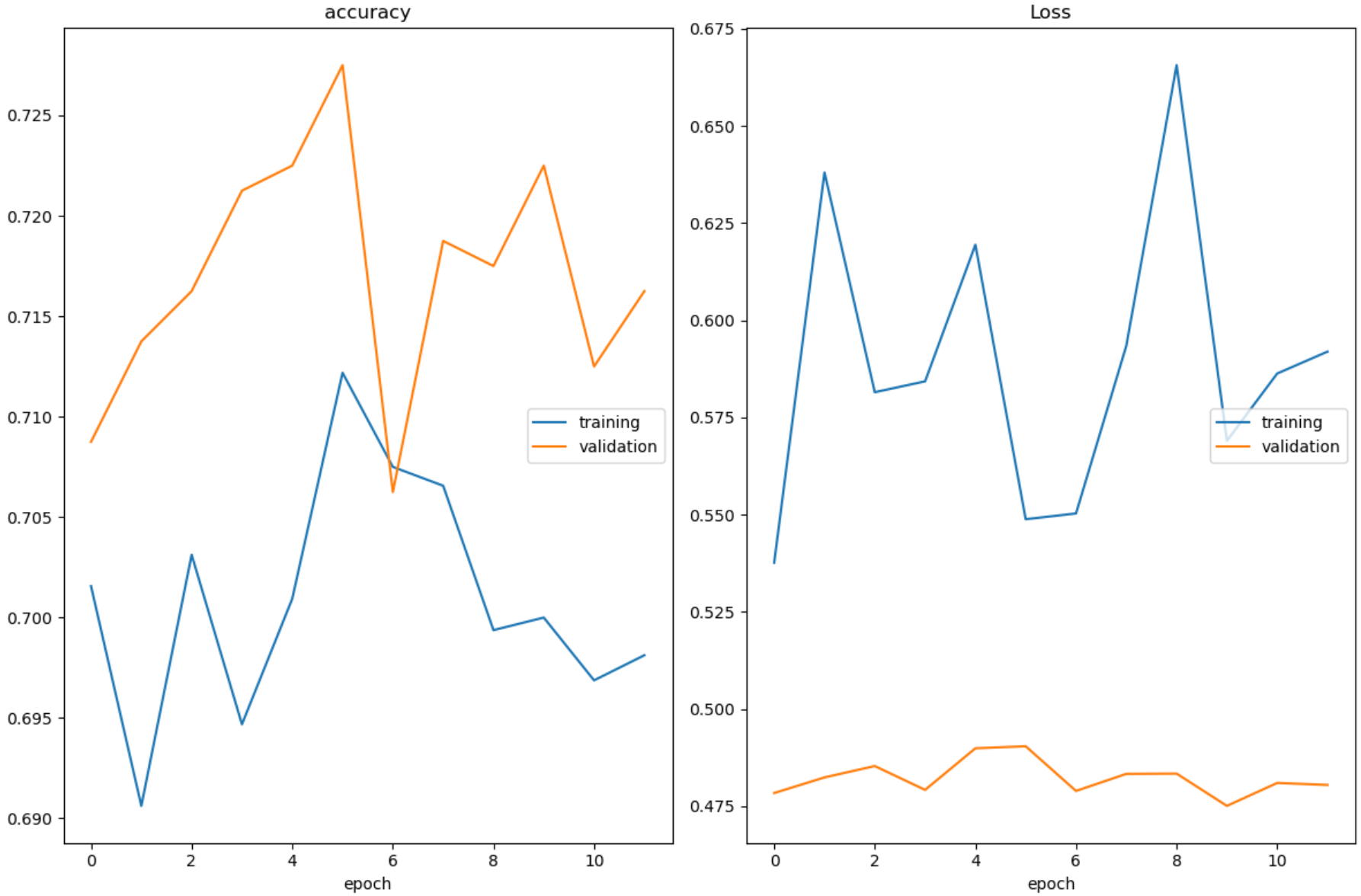
In [27]:

```
checkpoint = ModelCheckpoint('model_weights.hhs', monitor='val_loss',
                             save_weights_only=True, mode='min', verbose = 0)

callbacks = [PlotLossesCallback(), checkpoint]

batch_size = 32

history = model.fit(datagen_train.flow(x_train, y_train, batch_size=batch_size,
                                       shuffle=True),
                    steps_per_epoch = len(x_train) // batch_size,
                    validation_data = datagen_val.flow(x_val, y_val, batch_size=batch_size,
                                       shuffle=True),
                    epochs = 12,
                    callbacks=callbacks)
```


accuracy
0.725
0.720
0.715
0.710
0.705
0.700
0.695
0.690
0
2
4
6
8
10
epoch
training
validation
Loss
0.675
0.650
0.625
0.600
0.575
0.550
0.525
0.500
0.475
0
2
4
6
8
10
epoch
training
validation

```
accuracy
        training                 (min:    0.691, max:    0.712, cur:    0.698)
        validation               (min:    0.706, max:    0.728, cur:    0.716)
Loss
        training                 (min:    0.538, max:    0.666, cur:    0.592)
        validation               (min:    0.475, max:    0.490, cur:    0.480)
100/100 [==============================] - 102s 1s/step - loss: 0.5919 - accuracy: 0.6981 - val_loss: 0.4805 - val_a
ccuracy: 0.7163
```

9.Evaluate the Model

►Use the model.evaluate function to evaluate the accuracy

In [28]:
```
1 model.evaluate(x_val, y_val)
```

```
25/25 [==============================] - 3s 104ms/step - loss: 0.5685 - accuracy: 0.6712
```

Out[28]:
```
[0.5685462355613708, 0.6712499856948853]
```

▶Print a Classification Report and the accuracy score (classification accuracy)

In [29]:
```
y_true = np.argmax(y_val, 1)
y_pred = np.argmax(model.predict(x_val), 1)

print(metrics.classification_report(y_true, y_pred))
```

```
25/25 [==============================] - 3s 108ms/step
              precision    recall  f1-score   support

           0       0.63      0.99      0.77       200
           1       0.48      0.66      0.55       200
           2       0.57      0.04      0.07       200
           3       1.00      1.00      1.00       200

    accuracy                           0.67       800
   macro avg       0.67      0.67      0.60       800
weighted avg       0.67      0.67      0.60       800
```

▶Display a Confusion Matrix to evaluate the performance of the model

In [33]:
```
y_true = np.argmax(y_val, 1)
y_pred = np.argmax(model.predict(x_val), 1)

conf_matrix = metrics.confusion_matrix(y_true, y_pred)
print(metrics.confusion_matrix(y_true, y_pred))
```

```
25/25 [==============================] - 3s 105ms/step
[[198   2   0   0]
 [ 63 131   6   0]
 [ 52 140   8   0]
 [  0   0   0 200]]
```

In [ ]:
```
1
```

----------Thank you----------

In [ ]:
```
1
```