# Classify Radio Signals from Outer Space with Keras



## Task 1: Import Libraries

In [1]:

```
from livelossplot.tf_keras import PlotLossesCallback
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf

from sklearn.metrics import confusion_matrix
from sklearn import metrics

import numpy as np
np.random.seed(42)
import warnings;warnings.simplefilter('ignore')
%matplotlib inline
print('Tensorflow version:', tf.__version__)
```

```
Tensorflow version: 2.1.0
```

## Task 2: Load and Preprocess SETI Data

In [2]:

```
train_images = pd.read_csv('dataset/train/images.csv', header = None)
train_labels = pd.read_csv('dataset/train/labels.csv', header = None)

val_images = pd.read_csv('dataset/validation/images.csv', header = None)
val_labels = pd.read_csv('dataset/validation/labels.csv', header = None)
```

In [3]:

```
#shows pixel intensitiy values and they have been normalized (0, 1)
train_images.head(3)
```

Out[3]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.631373 | 0.623529 | 0.713726 | 0.705882 | 0.658824 | 0.666667 | 0.654902 | 0.635294 | 0.647059 | 0.70588 |
| **1** | 0.725490 | 0.752941 | 0.749020 | 0.701961 | 0.690196 | 0.721569 | 0.709804 | 0.745098 | 0.654902 | 0.72156 |
| **2** | 0.717647 | 0.701961 | 0.713726 | 0.733333 | 0.705882 | 0.717647 | 0.725490 | 0.682353 | 0.717647 | 0.67451 |

3 rows × 8192 columns

In [4]:

```
#each image is labeled from 0-3 showing different types of signals
#0 is squiggle
#1 narrow
#2 noise
#3 narrow band drd
train_labels.head(3)
```

Out[4]:

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| **0** | 1.0 | 0.0 | 0.0 | 0.0 |
| **1** | 1.0 | 0.0 | 0.0 | 0.0 |

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| **2** | 1.0 | 0.0 | 0.0 | 0.0 |

In [5]:

```
print('Training set shape:', train_images.shape, train_labels.shape)
print('Validation set shape:', val_images.shape, val_labels.shape)
#each training input has 8192 pixels, but it's actually distributed by a 64 width by 128
```

```
Training set shape: (3200, 8192) (3200, 4)
Validation set shape: (800, 8192) (800, 4)
```

In [6]:

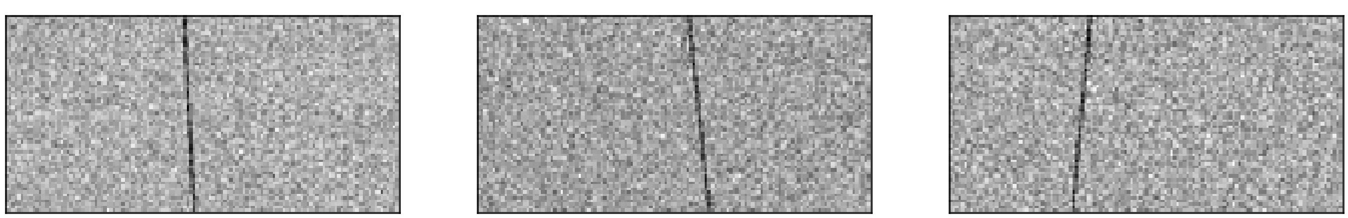
```
#reshape images for training and validation
#convert train_images to numpy array using .values to return n dimensional numpy array
#3200 # of training examples
#width = 64, height = 128, channel information (grayscale) = 1
x_train = train_images.values.reshape(3200, 64, 128, 1)
#800 validation samples
x_val = val_images.values.reshape(800, 64, 128, 1)

#reshape images labels for our image classificaiton model to accept as input
y_train = train_labels.values
y_val = val_labels.values
```

## Task 3: Plot 2D Spectrograms

In [7]:

```
#convert numpy array into 2d spectrograms
plt.figure(0, figsize = (12,12))
for i in range(1,4):
    plt.subplot(1, 3, i)
    #remove addional axis using np.squeeze
    img = np.squeeze(x_train[np.random.randint(0, x_train.shape[0])])
    plt.xticks([])
    plt.yticks([])
    #cmap = 'gray' if you want to see images in gray scale. this data set is in gray scal
    plt.imshow(img, cmap = 'gray')
```

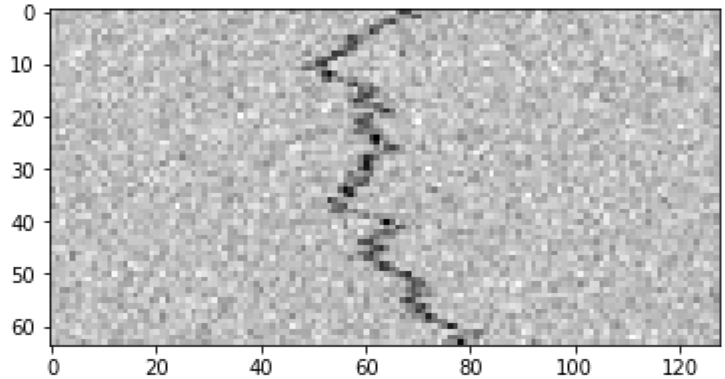


In [8]:

```
#view image in gray scale because we can't see the shape of the line
plt.imshow(np.squeeze(x_train[3]), cmap = 'gray')
```

Out[8]:

```
<matplotlib.image.AxesImage at 0x7f0505dc8e10>
```



In [ ]:

```

```

# Task 4: Create Training and Validation Data Generators

In [9]:

```python
#impoft image generator helper function to flip images horizontally
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen_train = ImageDataGenerator(horizontal_flip = True)

#fit data generator so it learns statistics and properties about the training images
datagen_train.fit(x_train)
datagen_val = ImageDataGenerator(horizontal_flip = True)
datagen_val.fit(x_val)

#no need to normalize because our data is already normalized
```

In [ ]:

```

```

# Task 5: Creating the CNN Model

In [10]:

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

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

In [11]:

```python
#CNN is a feed foward neural network consisting of multiple layers of neurons that have l
#network has multiple layers: convolution, maxpooling for downsampling, dropout layer for
#in each layer, small neurons process portions of the input images, and the outputs of th
#process repeated for each layer
#CNN able to break down complex patters in series of images into series of simpler patter

# Initialising the CNN
model = Sequential()

# 1st Convolution
#32 filter maps, fiter of 5 x 5
model.add(Conv2D(32, (5,5), padding = 'same', input_shape = (64, 128, 1)))
model.add(BatchNormalization())
#relu nonlinearity, works well with CNN
model.add(Activation('relu'))
#                     a form of nonlinear downsampling
```

```
model.add(MaxPooling2D(pool_size = (2,2)))
#regularization
model.add(Dropout(0.25))

# 2nd Convolution layer
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(0.25))

# Flattening
model.add(Flatten())

# Fully connected layer
model.add(Dense(1024))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.4))

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

In [ ]:

```

```

# Task 6: Learning Rate Scheduling and Compile the Model

In [12]:

```
initial_learning_rate = 0.005
#decay learning rate
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 [13]:

```
model.compile(optimizer= optimizer, loss = 'categorical_crossentropy', metrics = ['accura
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 64, 128, 32)       832       
_________________________________________________________________
batch_normalization (BatchNo (None, 64, 128, 32)       128       
_________________________________________________________________
activation (Activation)      (None, 64, 128, 32)       0         
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 64, 32)        0         
_________________________________________________________________
dropout (Dropout)            (None, 32, 64, 32)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 64, 64)        51264     
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 64, 64)        256       
_________________________________________________________________
activation_1 (Activation)    (None, 32, 64, 64)        0         
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 32, 64)        0         
_________________________________________________________________
```

```
dropout_1 (Dropout)          (None, 16, 32, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 32768)             0
_________________________________________________________________
dense (Dense)                (None, 1024)              33555456
_________________________________________________________________
batch_normalization_2 (Batch (None, 1024)              4096
_________________________________________________________________
activation_2 (Activation)    (None, 1024)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 4100
=================================================================
Total params: 33,616,132
Trainable params: 33,613,892
Non-trainable params: 2,240
_________________________________________________________________
```

In [ ]:

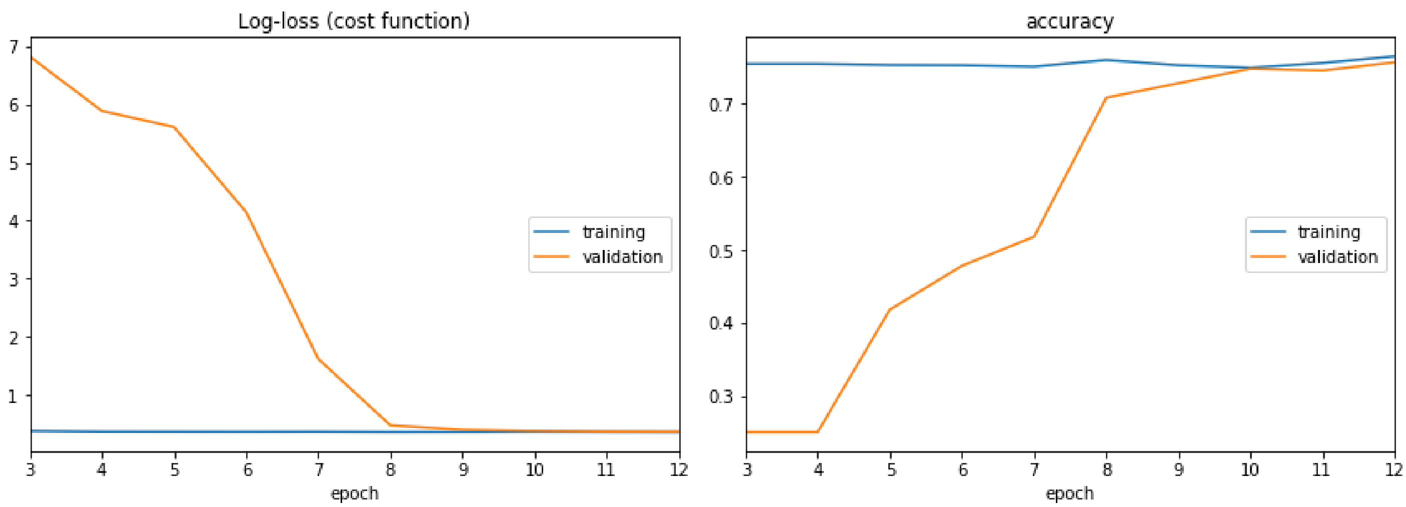
## Task 7: Training the Model

In [19]:

```
#defining callbacks
checkpoint = ModelCheckpoint('model_weight.h5', monitor='val_loss', save_weights_only=Tru
                             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 = T
    validation_steps = len(x_val) // batch_size,
    epochs = 12,
    callbacks = callbacks
)
```




```
Log-loss (cost function):
training   (min:    0.367, max:    0.597, cur:    0.367)
validation (min:    0.372, max:    6.822, cur:    0.372)

accuracy:
training   (min:    0.686, max:    0.764, cur:    0.764)
validation (min:    0.250, max:    0.756, cur:    0.756)
100/100 [==============================] - 8s 83ms/step - loss: 0.3673 - accuracy: 0.7644
_20 - val_accuracy: 0.7563
```

Loading [MathJax]/extensions/Safe.js

In [ ]:

```

```

# Task 8: Model Evaluation

In [20]:

```
model.evaluate(x_val, y_val)
```

```
800/800 [==============================] - 1s 853us/sample - loss: 0.3726 - accuracy: 0.75
50
```

Out[20]:

```
[0.3726379420934245, 0.755]
```

In [21]:

```
from sklearn.metrics import confusion_matrix
from sklearn import metrics
import seaborn as sns

y_true = np.argmax(y_val, 1)
y_pred = np.argmax(model.predict(x_val), 1)
print(metrics.classification_report(y_true, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       200
           1       0.53      0.23      0.33       200
           2       0.51      0.81      0.63       200
           3       1.00      1.00      1.00       200

    accuracy                           0.76       800
   macro avg       0.76      0.76      0.74       800
weighted avg       0.76      0.76      0.74       800
```

In [22]:

```
print('Classification accuracy: %0.6f' % metrics.accuracy_score(y_true, y_pred))
```

```
Classification accuracy: 0.755000
```

In [23]:

```
labels = ["squiggle", "narrowband", "noise", "narrowbanddrd"]
```

In [ ]:

```

```