Age and Gender Detection from Images

Team:

- Siva Yogitha Mokkapati
- Venkat Nihaal Akula

Course: DAAN 897– Deep Learning (Spring II, 2020)

Problem Statement

- Predicting the age and gender from facial images.
- **Keywords:** Classfication, Prediction, Deep Learning, emotion, Neural networks

Data Collection

- Source(url):(https://www.kaggle.com/jangedoo/utkface-new (https://www.kaggle.com/jangedoo/utkface-new)).
- Short Description : The dataset is taken from Kaggle. The dataset has approximately 23000 records with age and gender details as part of the filename
- Keywords: Age, Gender,images, prediction

Required packages

Install the required packages - Most of packages are already installed. The syntax for installing packages is pip install package name.

Next the installed package needs to be imported. The syntax is "import package name" or from "package" import "module name". The packages that are required are:

In [2]:
```
import numpy as np
import pandas as pd
import math
import numpy as np
import pandas as pd
import os
import cv2
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report
import tensorflow as tf
from tensorflow.keras import optimizers
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Conv2D, MaxPooling2D, Dropou
from tensorflow.keras.layers import Dropout, BatchNormalization, LeakyReLU, Acti
from tensorflow.keras.callbacks import Callback, EarlyStopping, ReduceLROnPlatea
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
```

Load the dataset and check number of files and a sample file

In [3]:
```
path = "C:/Users/nihal/Desktop/DL_2/UTKFace/"
files = os.listdir(path)
size = len(files)
print("Total samples:",size)
print(files[0])
```

```
Total samples: 23708
100_0_0_20170112213500903.jpg.chip.jpg
```

Read the image data and split age and gender values from its name

In [4]:
```
images = []
age = []
gender = []
for file in files:
    image = cv2.imread(path+file,0)
    #image = image.reshape((image.shape[0],image.shape[1],3))
    image = np.resize(image,(48,48,3))
    images.append(image)
    split_var = file.split('_')
    age.append(split_var[0])
    gender.append(int(split_var[1]))
```

Create Classes Split based on Age Limits

In [5]:
```
age_groups = []
for i in age:
    i = int(i)
    if i < 18:
        age_groups.append(0)
    if (i>=18) and (i<35):
        age_groups.append(1)
    if (i>=35) and (i<50):
        age_groups.append(2)
    if (i>=50) and (i<65):
        age_groups.append(3)
    if (i>=65):
        age_groups.append(4)
```

Data Preprocessing

- The images are resized into required format of (48,48,3) where 3 is the RGB format.
- Input images are squeezed to remove 1D images
- The pixels are normalized by dividing with 255.
- Age and Gender classes converted to categorical varaibles using label encoder.
- The data is spilitted into test and train.

Removing the single dimension entries for better prediction

In [6]:
```
images = np.squeeze(images)
images.shape
```

Out[6]:
```
(23708, 48, 48, 3)
```

Normalization Of the Images

In [7]:
```
images = images.astype('float32')/ 255
```

Convert Age and Gender Labels into Categorical Format

In [8]:
```
import keras
classes = keras.utils.to_categorical(age_groups,5)
gender = keras.utils.to_categorical(gender,2)
```

```
Using TensorFlow backend.
```

Split the data to train and validation in the ration 80:20

In [9]:
```
#Splitting the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(images, classes, test_size=0
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
print(X_train, X_test, y_train, y_test)
```

```
(18966, 48, 48, 3) (4742, 48, 48, 3) (18966, 5) (4742, 5)
[[[[0.1764706  0.17254902 0.16862746]
   [0.16470589 0.16470589 0.16470589]
   [0.16470589 0.16862746 0.15686275]
   ...
   [0.79607844 0.8156863  0.8235294 ]
   [0.8392157  0.84313726 0.84313726]
   [0.8392157  0.84313726 0.84705883]]

  [[0.85490197 0.8627451  0.8784314 ]
   [0.8901961  0.8980392  0.9019608 ]
   [0.8980392  0.8980392  0.9098039 ]
   ...
   [0.4        0.40392157 0.40392157]
   [0.40784314 0.4117647  0.41568628]
   [0.41568628 0.41960785 0.41960785]]

  [[0.42352942 0.42745098 0.42745098]
   [0.43529412 0.4392157  0.44313726]
```

Get the image width, height, depth and number of classes to use them while building models

In [10]:
```
img_width = X_test.shape[1]
img_height = X_test.shape[2]
img_depth = X_test.shape[3]
num_classes = y_test.shape[1]
print(img_width,img_height,img_depth,num_classes)
```

```
48 48 3 5
```

Methodology

1. To build deep neural networks and convolutional neural networks for emotion detection. To appl transfer learning (VGG) for training the model by making appropriate changes for the VGG model after importing it for emotion detection. To use a pre trained model for gender and age detection.
2. Deep Neural Networks you used in the project
    - ConvNet
        - A convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery(source Wikipedia)
3. Keywords
   Keywords: supervised learning, classification, prediction, neural networks.

Age Classification

Model fitting / Validation / Evaluation for age

1. Model

   6 CNN layers with 32, 64, 128 neurons for each two layers and 2D max pooling layer after every 2 layers.

2. Model 4

   Activation function relu, kernel initializer “he_normal”, padding “same”, kernel size (5,5) for first layer and (3,3) for the rest of the layers, dropout 0.4, batch normalization after each layer. Total 6 CNN layers and 2 dense layers at the end.

   2D max pooling of (2,2) after every 2 layers. Data augmentation (rotation_range=15,width_shift_range=0.15, height_shift_range=0.15,shear_range=0.15,zoom_range=0.15,horizontal_flip=True)

3. Model 5 Generating the first model with less convolution Layers for better accuracy

4. Model 6 Same as model 4 but the activation function is elu

5. Model 7 Building a simple CNN model with bigger kernel weights (96, 256, 384)

Optimal Age Detection Model: Model 4

- Optimizer: Adam
- Loss Function : Categorical Cross entropy
- Regularization: Drop out, Batch Normalization
- Output Activation Function: Softmax
- Activation Function: Relu
- Kernel size: (3,3) and (5,5)
- Kernel initializer: he_normal
- Pooling Layers: Max Pooling (2D)

First Model- A simple Convolution Neural Network Model

In [23]:
```
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(img_width, img_heigl
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(5, activation='softmax'))
model.summary()

model.compile(loss='categorical_crossentropy', optimizer=optimizers.Adam(learnin
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 46, 46, 32)        896
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 23, 23, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 21, 21, 64)        18496
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 10, 10, 64)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 2, 2, 128)         147584
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 1, 1, 128)         0
```

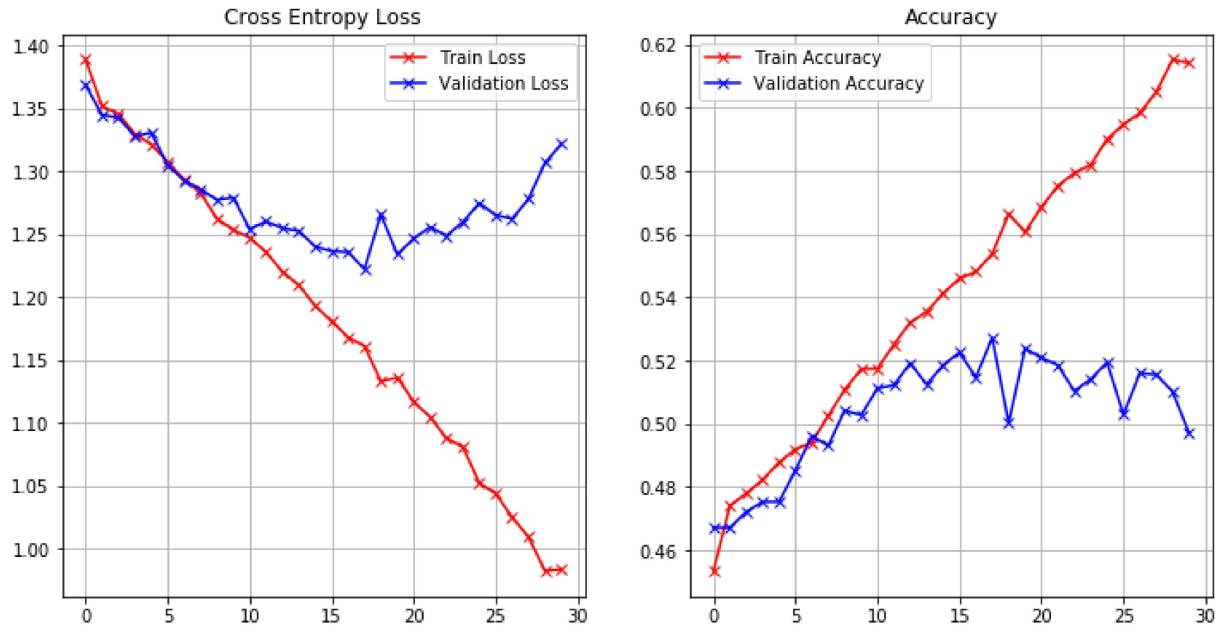
In [ ]:
```
#Now fit the model for 30 epcohs with bacth size
history = model.fit(X_train, y_train, batch_size = 210, epochs=30, verbose=1, va
```

```
Train on 18966 samples, validate on 4742 samples
Epoch 1/30
18966/18966 [==============================] - 24s 1ms/sample - loss: 0.9646 -
accuracy: 0.4920 - val_loss: 0.9330 - val_accuracy: 0.5063
Epoch 2/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.9429 -
accuracy: 0.5020 - val_loss: 0.9178 - val_accuracy: 0.5063
Epoch 3/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.9315 -
accuracy: 0.5025 - val_loss: 0.9105 - val_accuracy: 0.5063
Epoch 4/30
18966/18966 [==============================] - 21s 1ms/sample - loss: 0.9254 -
accuracy: 0.5023 - val_loss: 0.9153 - val_accuracy: 0.5063
Epoch 5/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.9254 -
accuracy: 0.5024 - val_loss: 0.9034 - val_accuracy: 0.5063
Epoch 6/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.9184 -
accuracy: 0.5028 - val_loss: 0.9031 - val_accuracy: 0.5063
Epoch 7/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.9181 -
accuracy: 0.5024 - val_loss: 0.9057 - val_accuracy: 0.5063
Epoch 8/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.9205 -
accuracy: 0.5022 - val_loss: 0.9092 - val_accuracy: 0.5063
Epoch 9/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.9154 -
accuracy: 0.5029 - val_loss: 0.9042 - val_accuracy: 0.5063
Epoch 10/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.9155 -
accuracy: 0.5032 - val_loss: 0.9018 - val_accuracy: 0.5063
Epoch 11/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.9179 -
accuracy: 0.5041 - val_loss: 0.9082 - val_accuracy: 0.5063
Epoch 12/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.9176 -
accuracy: 0.5034 - val_loss: 0.8977 - val_accuracy: 0.5063
Epoch 13/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.9126 -
accuracy: 0.5034 - val_loss: 0.8971 - val_accuracy: 0.5063
Epoch 14/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.9130 -
accuracy: 0.5028 - val_loss: 0.8974 - val_accuracy: 0.5158
Epoch 15/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.9093 -
accuracy: 0.5084 - val_loss: 0.8961 - val_accuracy: 0.5150
Epoch 16/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.9070 -
accuracy: 0.5095 - val_loss: 0.8929 - val_accuracy: 0.5186
Epoch 17/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.9063 -
accuracy: 0.5143 - val_loss: 0.8916 - val_accuracy: 0.5192
Epoch 18/30
```

```
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.9041 -
accuracy: 0.5132 - val_loss: 0.8898 - val_accuracy: 0.5259
Epoch 19/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.9031 -
accuracy: 0.5184 - val_loss: 0.8897 - val_accuracy: 0.5278
Epoch 20/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.9000 -
accuracy: 0.5209 - val_loss: 0.8888 - val_accuracy: 0.5226
Epoch 21/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.9007 -
accuracy: 0.5204 - val_loss: 0.8933 - val_accuracy: 0.5186
Epoch 22/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.8993 -
accuracy: 0.5218 - val_loss: 0.8805 - val_accuracy: 0.5358
Epoch 23/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.8992 -
accuracy: 0.5293 - val_loss: 0.8820 - val_accuracy: 0.5390
Epoch 24/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.8958 -
accuracy: 0.5294 - val_loss: 0.8802 - val_accuracy: 0.5481
Epoch 25/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.8939 -
accuracy: 0.5286 - val_loss: 0.8928 - val_accuracy: 0.5358
Epoch 26/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.8939 -
accuracy: 0.5328 - val_loss: 0.8821 - val_accuracy: 0.5405
Epoch 27/30
18966/18966 [==============================] - 22s 1ms/sample - loss: 0.8933 -
accuracy: 0.5330 - val_loss: 0.8791 - val_accuracy: 0.5449
Epoch 28/30
 1260/18966 [>.............................] - ETA: 21s - loss: 0.8928 - accura
cy: 0.5294
```

We can see that the accuracy on train is ---- and on validation is --- which indicates that the model is underfitting. Also, the train and the validation loss are very high. Hence we need to build a complex model.

In [23]:
```
fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 2, 1)
ax.plot(history.history["loss"], 'r-x', label="Train Loss")
ax.plot(history.history["val_loss"], 'b-x', label="Validation Loss")
ax.legend()
ax.set_title('Cross Entropy Loss')
ax.grid(True)

ax = fig.add_subplot(1, 2, 2)
ax.plot(history.history["accuracy"], 'r-x', label="Train Accuracy")
ax.plot(history.history["val_accuracy"], 'b-x', label="Validation Accuracy")
ax.legend()
ax.set_title('Accuracy')
ax.grid(True)
```



Second Model- Complex CNN Model.

In [11]:
```
model4 = Sequential()
model4.add(Conv2D(filters=64,kernel_size=(5,5),input_shape=(img_width, img_heigh
                      padding='same',kernel_initializer='he_normal'))
model4.add(BatchNormalization())
model4.add(Conv2D(
            filters=64,
            kernel_size=(5,5),
            activation='relu',
            padding='same',
            kernel_initializer='he_normal'))
model4.add(BatchNormalization())
model4.add(MaxPooling2D(pool_size=(2,2)))
model4.add(Dropout(0.4))
model4.add(Conv2D(
            filters=128,
            kernel_size=(3,3),
            activation='relu',
            padding='same',
            kernel_initializer='he_normal'))
model4.add(BatchNormalization())
model4.add(Conv2D(
            filters=128,
            kernel_size=(3,3),
            activation='relu',
            padding='same',
            kernel_initializer='he_normal'))
model4.add(BatchNormalization())
model4.add(MaxPooling2D(pool_size=(2,2)))
model4.add(Dropout(0.4))
model4.add(Conv2D(
            filters=256,
            kernel_size=(3,3),
            activation='relu',
            padding='same',
            kernel_initializer='he_normal'))
model4.add(BatchNormalization())
model4.add(Conv2D(
            filters=256,
            kernel_size=(3,3),
            activation='relu',
            padding='same',
            kernel_initializer='he_normal'))
model4.add(BatchNormalization())
model4.add(MaxPooling2D(pool_size=(2,2)))
model4.add(Dropout(0.5, name='dropout_3'))
model4.add(Flatten())
model4.add( Dense(128,activation='relu',kernel_initializer='he_normal'))
model4.add(BatchNormalization())
model4.add(Dropout(0.5))
model4.add(Dense(
            num_classes,
            activation='softmax'))
model4.summary()
model4.compile(loss='categorical_crossentropy', optimizer=optimizers.Adam(learni
```

```
history4 = model4.fit(X_train, y_train, epochs=35, verbose=1, validation_data=(X_
```

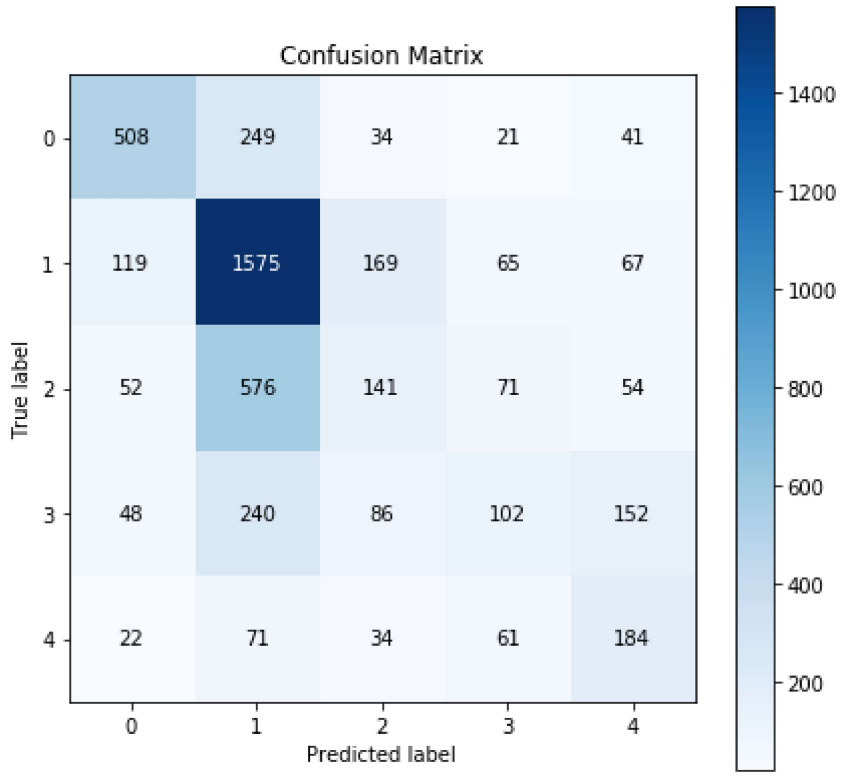
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 48, 48, 64)        4864
_________________________________________________________________
batch_normalization (BatchNo (None, 48, 48, 64)        256
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 48, 48, 64)        102464
_________________________________________________________________
batch_normalization_1 (Batch (None, 48, 48, 64)        256
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 24, 24, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 128)       73856
_________________________________________________________________
batch_normalization_2 (Batch (None, 24, 24, 128)       512
```

A Model has been generated with 69% Training Accuracy and 53% Validation Accuracy, After considering all the other models this is our best and optimal model

Generating the Confusion Matrix

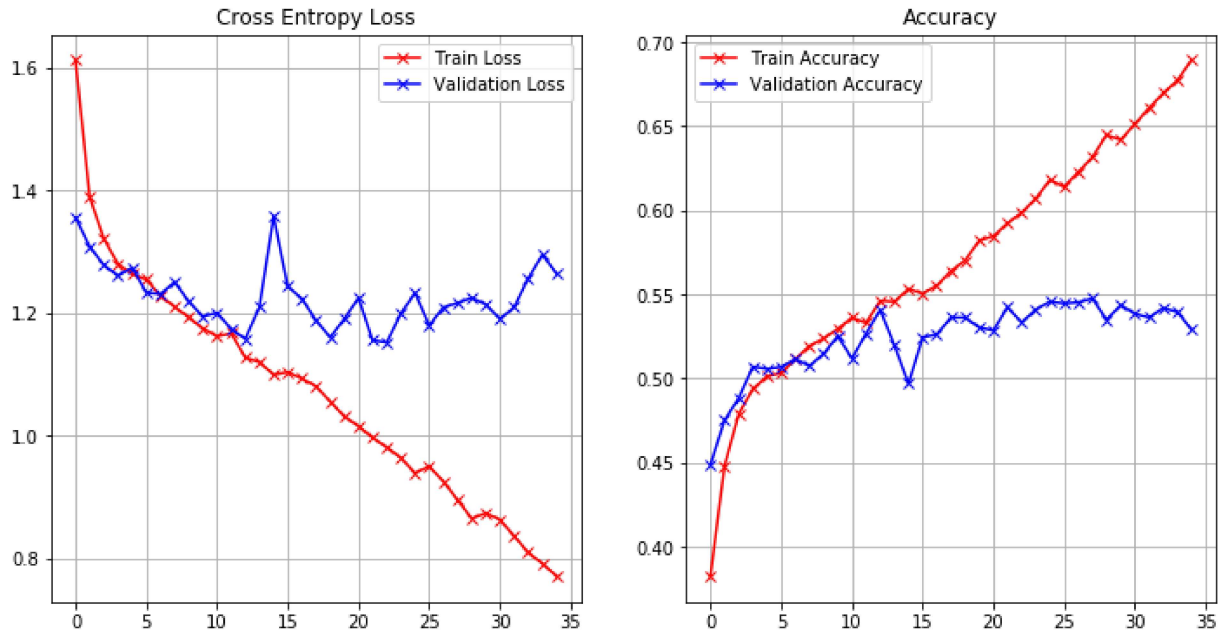
In [13]:
```
import scikitplot
y_valid = model4.predict_classes(X_test)
scikitplot.metrics.plot_confusion_matrix(np.argmax(y_test, axis=1), y_valid, fig

print(f'total wrong validation predictions: {np.sum(np.argmax(y_test, axis=1) !=
print(classification_report(np.argmax(y_test, axis=1), y_valid))
```

```
total wrong validation predictions: 2232

              precision    recall  f1-score   support

           0       0.68      0.60      0.63       853
           1       0.58      0.79      0.67      1995
           2       0.30      0.16      0.21       894
           3       0.32      0.16      0.22       628
           4       0.37      0.49      0.42       372

    accuracy                           0.53      4742
   macro avg       0.45      0.44      0.43      4742
weighted avg       0.49      0.53      0.50      4742
```



Plotting the Train Vs Validation Accuracy and Loss Graphs

In [14]:
```
fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 2, 1)
ax.plot(history4.history["loss"], 'r-x', label="Train Loss")
ax.plot(history4.history["val_loss"], 'b-x', label="Validation Loss")
ax.legend()
ax.set_title('Cross Entropy Loss')
ax.grid(True)

ax = fig.add_subplot(1, 2, 2)
ax.plot(history4.history["accuracy"], 'r-x', label="Train Accuracy")
ax.plot(history4.history["val_accuracy"], 'b-x', label="Validation Accuracy")
ax.legend()
ax.set_title('Accuracy')
ax.grid(True)
```



Saving the model

In [15]:
```
model4.save("C:/Users/nihal/Desktop/DL_2")
```

```
WARNING:tensorflow:From C:\Users\nihal\AppData\Roaming\Python\Python37\site-pac
kages\tensorflow_core\python\ops\resource_variable_ops.py:1786: calling BaseRes
ourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with
constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
INFO:tensorflow:Assets written to: C:/Users/nihal/Desktop/DL_2\assets
```

Third Model - Regenerating the first model with less convolution Layers for better accuracy

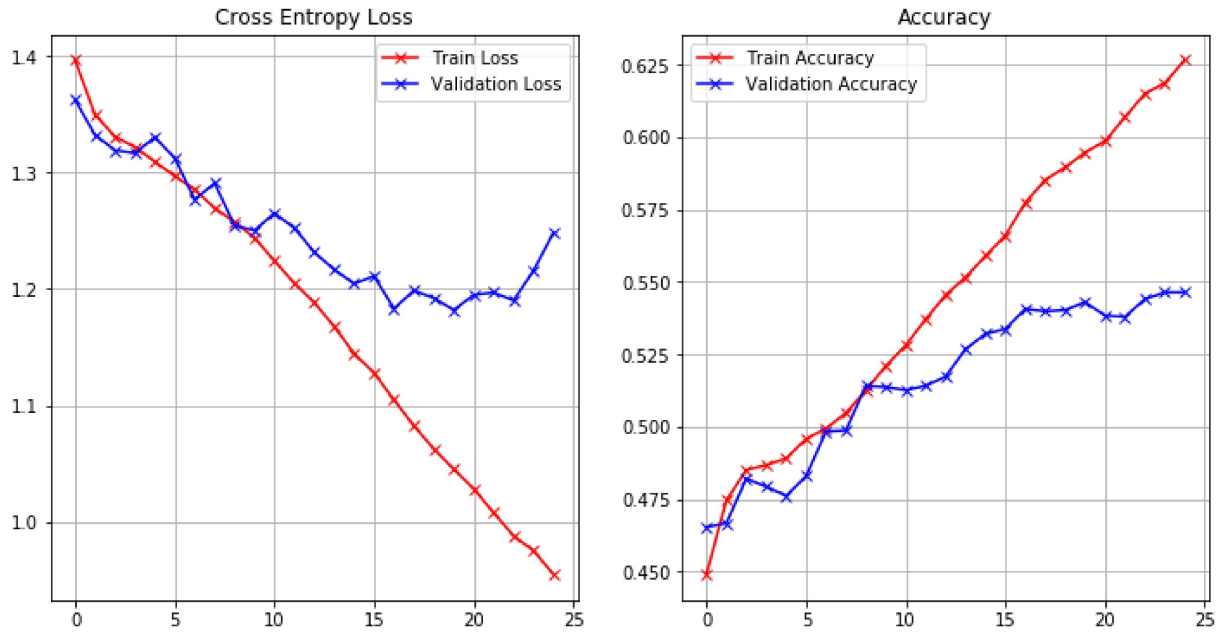
In [25]:
```
model5 = Sequential()

model5.add(Conv2D(32, (3, 3), activation='relu', input_shape=(img_width, img_hei
model5.add(Conv2D(64, (3, 3), activation='relu'))
model5.add(MaxPooling2D(pool_size=(2,2)))
model5.add(Conv2D(128, (3, 3), activation='relu'))
model5.add(MaxPooling2D(pool_size=(2,2)))
model5.add(Dropout(0.25))
model5.add(Flatten())

model5.add(Dense(128, activation='relu'))
model5.add(Dropout(0.3))
model5.add(Dense(64, activation='relu'))
model5.add(Dropout(0.3))
model5.add(Dense(32, activation='relu'))
model5.add(Dropout(0.3))
model5.add(Dense(5, activation='softmax'))
model5.summary()
model5.compile(loss='categorical_crossentropy', optimizer=optimizers.Adam(learni

history5 = model5.fit(X_train, y_train, epochs=25, verbose=1, validation_data=(X_
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 46, 46, 32)        896
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 44, 44, 64)        18496
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 22, 22, 64)        0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 20, 20, 128)       73856
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 10, 10, 128)       0
_________________________________________________________________
dropout_12 (Dropout)         (None, 10, 10, 128)       0
_________________________________________________________________
flatten_4 (Flatten)          (None, 12800)             0
_________________________________________________________________
dense_11 (Dense)             (None, 128)               1638528
```

Model with accuracy of 63% and validation accuracy of 55% and its an underfitting Model

Plotting the Train Vs Validation Accuracy and Loss Graphs

In [29]:
```
fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 2, 1)
ax.plot(history5.history["loss"], 'r-x', label="Train Loss")
ax.plot(history5.history["val_loss"], 'b-x', label="Validation Loss")
ax.legend()
ax.set_title('Cross Entropy Loss')
ax.grid(True)

ax = fig.add_subplot(1, 2, 2)
ax.plot(history5.history["accuracy"], 'r-x', label="Train Accuracy")
ax.plot(history5.history["val_accuracy"], 'b-x', label="Validation Accuracy")
ax.legend()
ax.set_title('Accuracy')
ax.grid(True)
```



Fourth Model - Recreating the second complex model using 'elu' as activation function

In [28]:
```
model6 = Sequential()
model6.add(Conv2D(filters=64,kernel_size=(5,5),input_shape=(img_width, img_height
                      padding='same',kernel_initializer='he_normal'))
model6.add(BatchNormalization())
model6.add(MaxPooling2D(pool_size=(2,2)))
model6.add(Dropout(0.4))
model6.add(Conv2D(
            filters=128,
            kernel_size=(3,3),
            activation='elu',
            padding='same',
            kernel_initializer='he_normal'))
model6.add(BatchNormalization())
model6.add(MaxPooling2D(pool_size=(2,2)))
model6.add(Dropout(0.4))
model6.add(Conv2D(
            filters=256,
            kernel_size=(3,3),
            activation='elu',
            padding='same',
            kernel_initializer='he_normal'))
model6.add(BatchNormalization())
model6.add(MaxPooling2D(pool_size=(2,2)))
model6.add(Dropout(0.4))
model6.add(Dropout(0.5, name='dropout_3'))
model6.add(Flatten())
model6.add( Dense(128,activation='elu',kernel_initializer='he_normal'))
model6.add(BatchNormalization())
model6.add(Dropout(0.5))
model6.add(Dense(
            5,
            activation='softmax'))
model6.summary()
model6.compile(loss='categorical_crossentropy', optimizer=optimizers.Adam(learni


history6 = model6.fit(X_train, y_train, epochs=25, verbose=1, validation_data=(X_
```

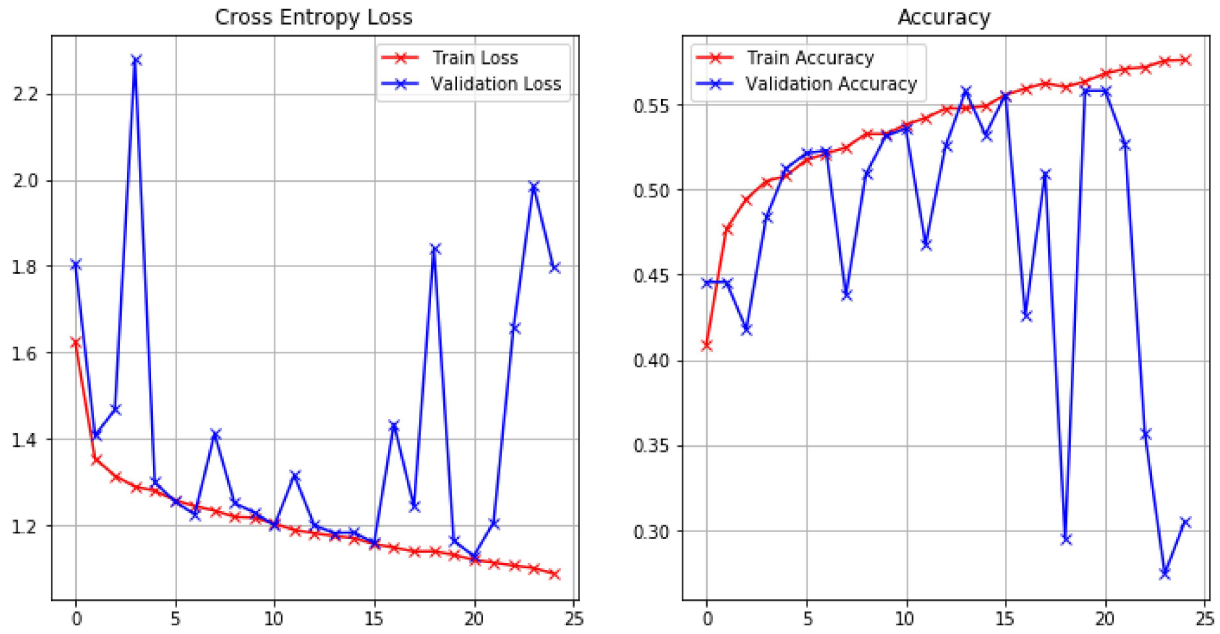
```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_21 (Conv2D)           (None, 48, 48, 64)        4864
_________________________________________________________________
batch_normalization_7 (Batch (None, 48, 48, 64)        256
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 24, 24, 64)        0
_________________________________________________________________
dropout_16 (Dropout)         (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 24, 24, 128)       73856
_________________________________________________________________
batch_normalization_8 (Batch (None, 24, 24, 128)       512
```

```
max_pooling2d_18 (MaxPooling (None, 12, 12, 128)       0
```

Model with accuracy of 58% and validation accuracy of 30%, its an underfitting model

Plotting the Train Vs Validation Accuracy and Loss Graphs

In [30]:
```
fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 2, 1)
ax.plot(history6.history["loss"], 'r-x', label="Train Loss")
ax.plot(history6.history["val_loss"], 'b-x', label="Validation Loss")
ax.legend()
ax.set_title('Cross Entropy Loss')
ax.grid(True)

ax = fig.add_subplot(1, 2, 2)
ax.plot(history6.history["accuracy"], 'r-x', label="Train Accuracy")
ax.plot(history6.history["val_accuracy"], 'b-x', label="Validation Accuracy")
ax.legend()
ax.set_title('Accuracy')
ax.grid(True)
```



Fifth Model - Building a simple CNN model with bigger kernel weights (96, 256, 384)

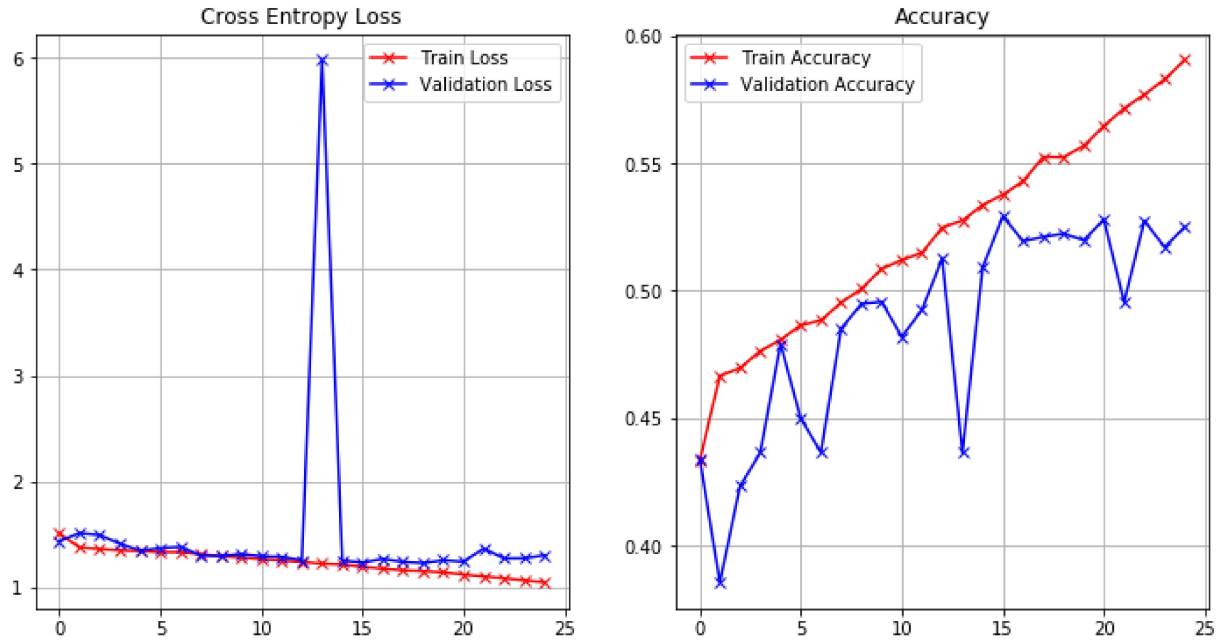
In [32]:
```
model7 = Sequential()

model7.add(Conv2D(96, (7, 7), activation='relu', input_shape=(img_width, img_hei
model7.add(MaxPooling2D(pool_size=(2, 2)))
model7.add(BatchNormalization())
model7.add(Conv2D(256, (5, 5), activation='relu'))
model7.add(MaxPooling2D(pool_size=(2, 2)))
model7.add(BatchNormalization())
model7.add(Conv2D(384, (3, 3), activation='relu'))
model7.add(MaxPooling2D(pool_size=(2, 2)))
model7.add(Flatten())
model7.add(Dense(512, activation='relu'))
model7.add(Dropout(0.3))
model7.add(Dense(128, activation='relu'))
model7.add(Dropout(0.3))
model7.add(Dense(5, activation='softmax'))
model7.summary()

model7.compile(loss='categorical_crossentropy', optimizer=optimizers.Adam(learni


history7 = model7.fit(X_train, y_train, epochs=25, verbose=1, validation_data=(X_
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_27 (Conv2D)           (None, 42, 42, 96)        14208
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 21, 21, 96)        0
_________________________________________________________________
batch_normalization_13 (Batc (None, 21, 21, 96)        384
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 17, 17, 256)       614656
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 8, 8, 256)         0
_________________________________________________________________
batch_normalization_14 (Batc (None, 8, 8, 256)         1024
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 6, 6, 384)         885120
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 3, 3, 384)         0
```

Model with accuracy of 59% and validation accuracy of 53%, which is an underfitting model

Plotting the Train Vs Validation Accuracy and Loss Graphs

In [33]:
```
fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 2, 1)
ax.plot(history7.history["loss"], 'r-x', label="Train Loss")
ax.plot(history7.history["val_loss"], 'b-x', label="Validation Loss")
ax.legend()
ax.set_title('Cross Entropy Loss')
ax.grid(True)

ax = fig.add_subplot(1, 2, 2)
ax.plot(history7.history["accuracy"], 'r-x', label="Train Accuracy")
ax.plot(history7.history["val_accuracy"], 'b-x', label="Validation Accuracy")
ax.legend()
ax.set_title('Accuracy')
ax.grid(True)
```



Gender Classification

Splitting the image and Gender classes data into Train and Test splits

In [9]:
```
X_train, X_test, y_train, y_test = train_test_split(images, gender, test_size=0.
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
print(X_train, X_test, y_train, y_test)
```

```
(18966, 48, 48, 3) (4742, 48, 48, 3) (18966, 2) (4742, 2)
[[[[0.14509805 0.14509805 0.14901961]
   [0.15294118 0.15294118 0.15294118]
   [0.14901961 0.14901961 0.13725491]
   ...
   [0.5019608  0.49019608 0.5176471 ]
   [0.53333336 0.5254902  0.5137255 ]
   [0.49019608 0.45490196 0.41960785]]

  [[0.36078432 0.32941177 0.28235295]
   [0.23529412 0.20392157 0.18039216]
   [0.16470589 0.15294118 0.15294118]
   ...
   [0.25882354 0.26666668 0.27058825]
   [0.2784314  0.2901961  0.29411766]
   [0.2901961  0.27450982 0.25882354]]

  [[0.23529412 0.22352941 0.21176471]
   [0.20784314 0.21176471 0.21176471]
```

Get the image width, height, depth and number of classes to use them while building models

In [10]:
```
img_width = X_test.shape[1]
img_height = X_test.shape[2]
img_depth = X_test.shape[3]
num_classes = y_test.shape[1]
print(img_width,img_height,img_depth,num_classes)
```

```
48 48 3 2
```

Model fitting / Validation / Evaluation for age

1. Modelg

   4 Convolution layers and 3 dense layers with max pooling after each layer and dropout after all the convolutional layers. Activation function for output layer is sigmoid and relu for other layers.

2. Modelg1 Simple CNN model with bigger kernel weights (96, 256, 384). 3 CNN layers and 3 Dense layers with max pooling and batch normalization after each CNN layer. Drop out is added after each dense layer. Sigmoid for output layer and relu for other layers.

3. Modelg2 Basic CNN model

Optimal Gender Detection Model :Model g1

- Optimizer: Adam
- Loss Function : Binary Cross entropy

- Regularization: Drop out, Batch Normalization
- Output Activation Function: Sigmoid
- Activation Function: Relu
- Kernel size: (3,3) and (5,5)
- Pooling Layers: Max Pooling (2D)

Creating a Simple CNN Model

In [21]:
```
modelg = Sequential()

modelg.add(Conv2D(32, (3, 3), activation='relu', input_shape=(img_width, img_hei
modelg.add(MaxPooling2D(pool_size=(2, 2)))
modelg.add(Conv2D(64, (3, 3), activation='relu'))
modelg.add(MaxPooling2D(pool_size=(2, 2)))
modelg.add(Conv2D(128, (3, 3), activation='relu'))
modelg.add(MaxPooling2D(pool_size=(2, 2)))
modelg.add(Conv2D(128, (3, 3), activation='relu'))
modelg.add(MaxPooling2D(pool_size=(2, 2)))
modelg.add(Dropout(0.3))
modelg.add(Flatten())
modelg.add(Dense(256, activation='relu'))
modelg.add(Dropout(0.3))
modelg.add(Dense(128, activation='relu'))
modelg.add(Dropout(0.3))
modelg.add(Dense(2, activation='sigmoid'))
modelg.summary()

modelg.compile(loss='binary_crossentropy', optimizer=optimizers.Adam(learning_ra

historyg = modelg.fit(X_train, y_train, epochs=25, verbose=1, validation_data=(X_
```

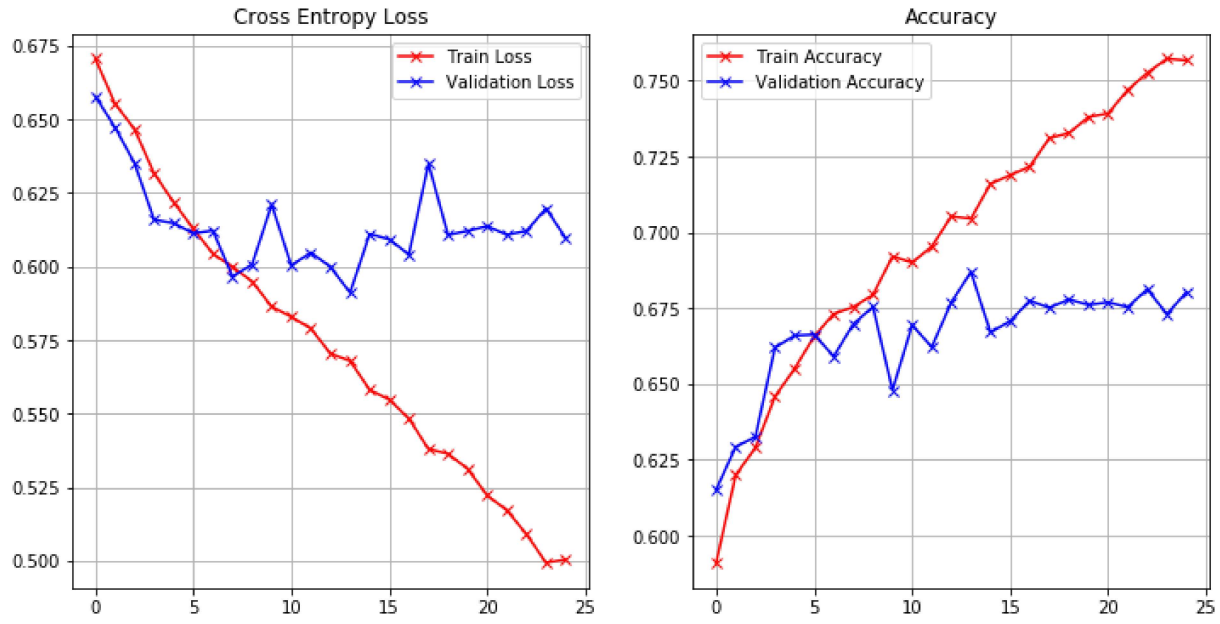
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 46, 46, 32)        896
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 23, 23, 32)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 21, 21, 64)        18496
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 10, 10, 64)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 8, 8, 128)         73856
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 2, 2, 128)         147584
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 1, 1, 128)         0
```

This model has accuracy of 76% and validation accuracy of 68%, which is a good but an

underfitting model

Plotting the Train Vs Validation Accuracy and Loss Graphs

In [22]:
```
fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 2, 1)
ax.plot(historyg.history["loss"], 'r-x', label="Train Loss")
ax.plot(historyg.history["val_loss"], 'b-x', label="Validation Loss")
ax.legend()
ax.set_title('Cross Entropy Loss')
ax.grid(True)

ax = fig.add_subplot(1, 2, 2)
ax.plot(historyg.history["accuracy"], 'r-x', label="Train Accuracy")
ax.plot(historyg.history["val_accuracy"], 'b-x', label="Validation Accuracy")
ax.legend()
ax.set_title('Accuracy')
ax.grid(True)
```



Building a simple CNN model with bigger kernel weights (96, 256, 384)

In [11]:
```
modelg1 = Sequential()

modelg1.add(Conv2D(96, (7, 7), activation='relu', input_shape=(img_width, img_he
modelg1.add(MaxPooling2D(pool_size=(2, 2)))
modelg1.add(BatchNormalization())
modelg1.add(Conv2D(256, (5, 5), activation='relu'))
modelg1.add(MaxPooling2D(pool_size=(2, 2)))
modelg1.add(BatchNormalization())
modelg1.add(Conv2D(384, (3, 3), activation='relu'))
modelg1.add(MaxPooling2D(pool_size=(2, 2)))
modelg1.add(Flatten())
modelg1.add(Dense(512, activation='relu'))
modelg1.add(Dropout(0.3))
modelg1.add(Dense(128, activation='relu'))
modelg1.add(Dropout(0.3))
modelg1.add(Dense(2, activation='sigmoid'))
modelg1.summary()

modelg1.compile(loss='binary_crossentropy', optimizer=optimizers.Adam(learning_r

historyg1 = modelg1.fit(X_train, y_train, epochs=20, verbose=1, validation_data=
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 42, 42, 96)        14208
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 21, 21, 96)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 21, 21, 96)        384
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 17, 17, 256)       614656
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 256)         0
_________________________________________________________________
batch_normalization_1 (Batch (None, 8, 8, 256)         1024
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 6, 6, 384)         885120
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 384)         0
_________________________________________________________________
flatten (Flatten)            (None, 3456)              0
_________________________________________________________________
dense (Dense)                (None, 512)               1769984
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               65664
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 258
```

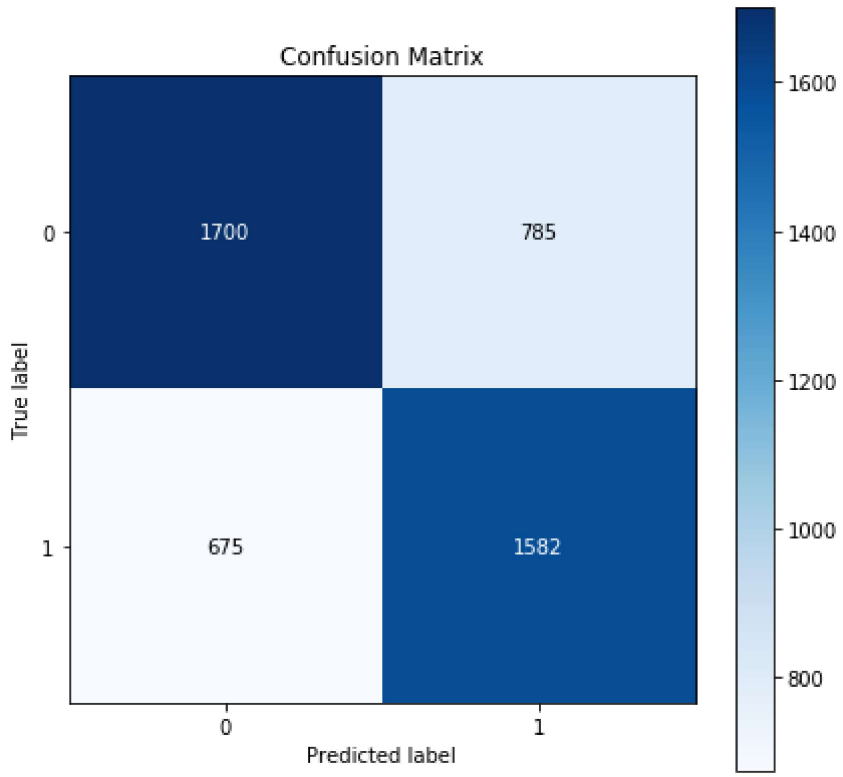
```
=================================================================
Total params: 3,351,298
Trainable params: 3,350,594
Non-trainable params: 704
_________________________________________________________________
Train on 18966 samples, validate on 4742 samples
Epoch 1/20
18966/18966 [==============================] - 254s 13ms/sample - loss: 0.7206
- accuracy: 0.5668 - val_loss: 0.6825 - val_accuracy: 0.6054
Epoch 2/20
18966/18966 [==============================] - 178s 9ms/sample - loss: 0.6702 -
accuracy: 0.6020 - val_loss: 0.7145 - val_accuracy: 0.4948
Epoch 3/20
18966/18966 [==============================] - 173s 9ms/sample - loss: 0.6581 -
accuracy: 0.6152 - val_loss: 0.6604 - val_accuracy: 0.6379
Epoch 4/20
18966/18966 [==============================] - 175s 9ms/sample - loss: 0.6448 -
accuracy: 0.6400 - val_loss: 0.7535 - val_accuracy: 0.5636
Epoch 5/20
18966/18966 [==============================] - 174s 9ms/sample - loss: 0.6293 -
accuracy: 0.6535 - val_loss: 0.6284 - val_accuracy: 0.6484
Epoch 6/20
18966/18966 [==============================] - 177s 9ms/sample - loss: 0.6163 -
accuracy: 0.6685 - val_loss: 0.6262 - val_accuracy: 0.6571
Epoch 7/20
18966/18966 [==============================] - 177s 9ms/sample - loss: 0.6027 -
accuracy: 0.6774 - val_loss: 0.6049 - val_accuracy: 0.6583
Epoch 8/20
18966/18966 [==============================] - 183s 10ms/sample - loss: 0.5925
- accuracy: 0.6867 - val_loss: 0.6098 - val_accuracy: 0.6770
Epoch 9/20
18966/18966 [==============================] - 186s 10ms/sample - loss: 0.5748
- accuracy: 0.7004 - val_loss: 0.6107 - val_accuracy: 0.6559
Epoch 10/20
18966/18966 [==============================] - 196s 10ms/sample - loss: 0.5662
- accuracy: 0.7067 - val_loss: 0.6291 - val_accuracy: 0.6720
Epoch 11/20
18966/18966 [==============================] - 189s 10ms/sample - loss: 0.5518
- accuracy: 0.7208 - val_loss: 0.5888 - val_accuracy: 0.6953
Epoch 12/20
18966/18966 [==============================] - 198s 10ms/sample - loss: 0.5387
- accuracy: 0.7301 - val_loss: 0.5729 - val_accuracy: 0.6943
Epoch 13/20
18966/18966 [==============================] - 193s 10ms/sample - loss: 0.5243
- accuracy: 0.7375 - val_loss: 0.5959 - val_accuracy: 0.7001
Epoch 14/20
18966/18966 [==============================] - 189s 10ms/sample - loss: 0.5168
- accuracy: 0.7452 - val_loss: 0.5540 - val_accuracy: 0.7082
Epoch 15/20
18966/18966 [==============================] - 204s 11ms/sample - loss: 0.4931
- accuracy: 0.7556 - val_loss: 0.5624 - val_accuracy: 0.7140
Epoch 16/20
18966/18966 [==============================] - 202s 11ms/sample - loss: 0.4786
- accuracy: 0.7689 - val_loss: 0.5538 - val_accuracy: 0.7160
Epoch 17/20
18966/18966 [==============================] - 188s 10ms/sample - loss: 0.4547
- accuracy: 0.7823 - val_loss: 0.6485 - val_accuracy: 0.6271
```

```
Epoch 18/20
18966/18966 [==============================] - 199s 11ms/sample - loss: 0.4331
- accuracy: 0.7908 - val_loss: 0.5769 - val_accuracy: 0.7009
Epoch 19/20
18966/18966 [==============================] - 205s 11ms/sample - loss: 0.4076
- accuracy: 0.8064 - val_loss: 0.5888 - val_accuracy: 0.7042
Epoch 20/20
18966/18966 [==============================] - 191s 10ms/sample - loss: 0.3843
- accuracy: 0.8202 - val_loss: 0.7208 - val_accuracy: 0.6920
```

In [12]:
```
import scikitplot
y_valid = modelg1.predict_classes(X_test)
scikitplot.metrics.plot_confusion_matrix(np.argmax(y_test, axis=1), y_valid, fig

print(f'total wrong validation predictions: {np.sum(np.argmax(y_test, axis=1) !=
print(classification_report(np.argmax(y_test, axis=1), y_valid))
```

```
total wrong validation predictions: 1460
```

	precision	recall	f1-score	support
0	0.72	0.68	0.70	2485
1	0.67	0.70	0.68	2257
accuracy			0.69	4742
macro avg	0.69	0.69	0.69	4742
weighted avg	0.69	0.69	0.69	4742



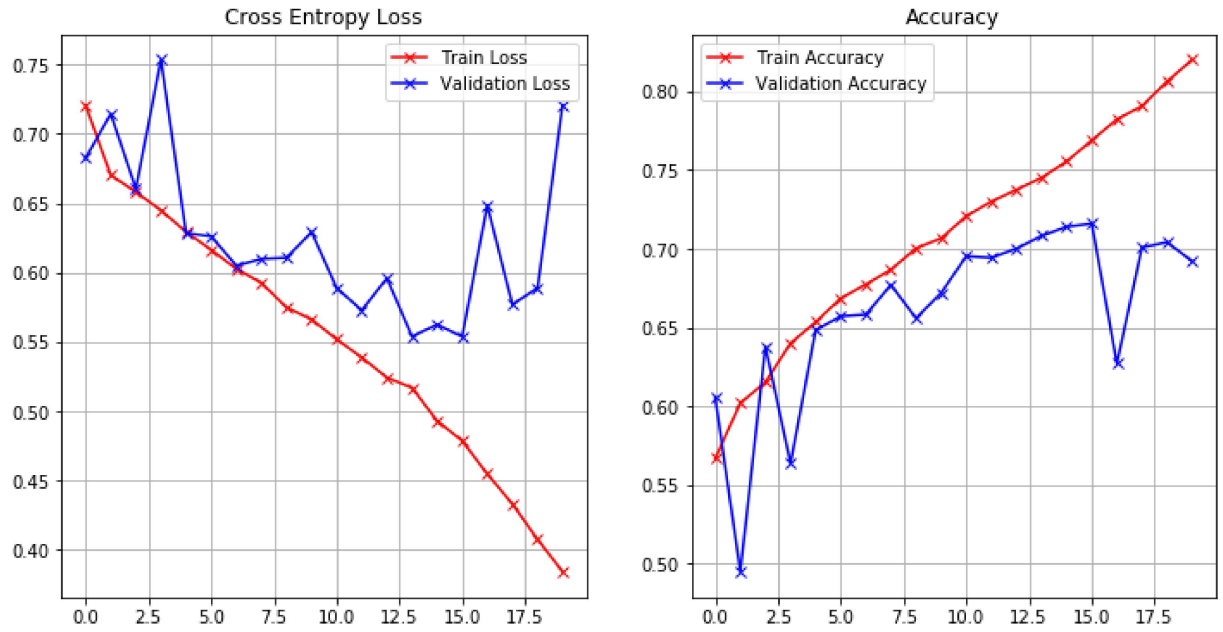
In [14]:
```
modelg1.save("C:/Users/nihal/Desktop/DL_2")
```

```
WARNING:tensorflow:From C:\Users\nihal\AppData\Roaming\Python\Python37\site-pac
kages\tensorflow_core\python\ops\resource_variable_ops.py:1786: calling BaseRes
ourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with
constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
INFO:tensorflow:Assets written to: C:/Users/nihal/Desktop/DL_2\assets
```

The model produced has training accuracy of 83% and validation accuracy of 71%, its an overfitting model and also huge gap in loss values

Plotting the Train Vs Validation Accuracy and Loss Graphs

In [13]:
```
fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 2, 1)
ax.plot(historyg1.history["loss"], 'r-x', label="Train Loss")
ax.plot(historyg1.history["val_loss"], 'b-x', label="Validation Loss")
ax.legend()
ax.set_title('Cross Entropy Loss')
ax.grid(True)

ax = fig.add_subplot(1, 2, 2)
ax.plot(historyg1.history["accuracy"], 'r-x', label="Train Accuracy")
ax.plot(historyg1.history["val_accuracy"], 'b-x', label="Validation Accuracy")
ax.legend()
ax.set_title('Accuracy')
ax.grid(True)
```



In [18]:
```
modelg2 = Sequential()

modelg2.add(Conv2D(32, (5, 5), activation='relu', input_shape=(img_width, img_he
modelg2.add(MaxPooling2D(pool_size=(2, 2)))
modelg2.add(BatchNormalization())
modelg2.add(Conv2D(64, (3, 3), activation='relu'))
modelg2.add(MaxPooling2D(pool_size=(2, 2)))
modelg2.add(BatchNormalization())
modelg2.add(Conv2D(128, (3, 3), activation='relu'))
modelg2.add(MaxPooling2D(pool_size=(2, 2)))
modelg2.add(Flatten())
modelg2.add(Dense(128, activation='relu'))
modelg2.add(Dropout(0.3))
modelg2.add(Dense(2, activation='sigmoid'))
modelg2.summary()

modelg2.compile(loss='binary_crossentropy', optimizer=optimizers.Adam(learning_r

historyg2 = modelg2.fit(X_train, y_train, epochs=11, verbose=1, validation_data=
```

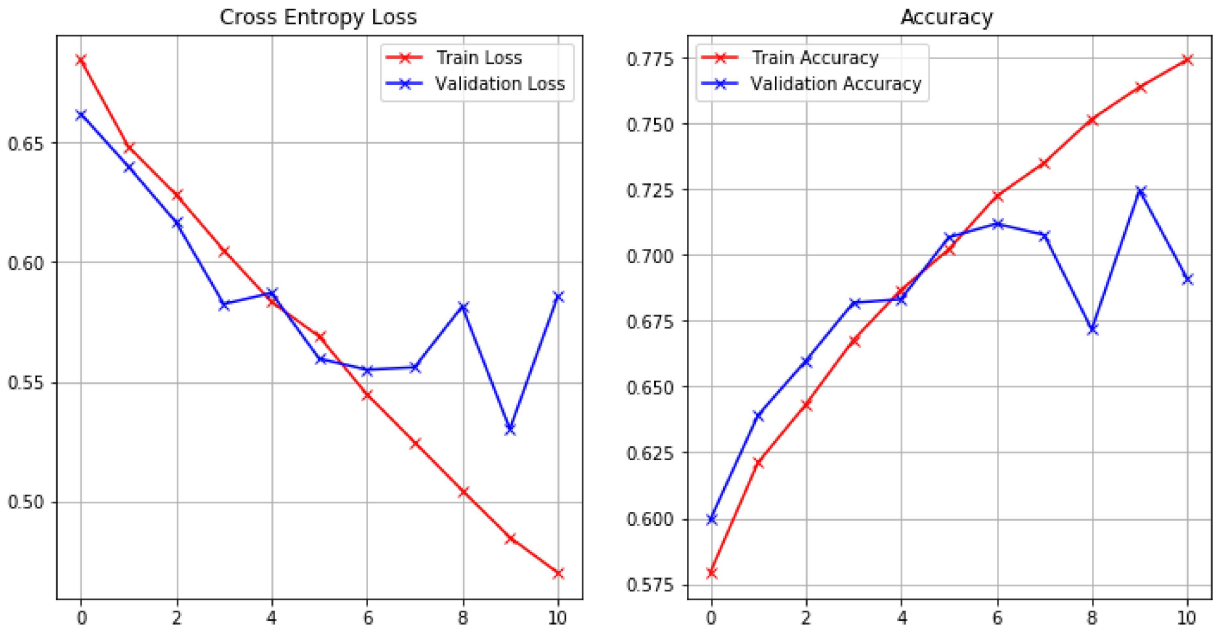
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 44, 44, 32)        2432
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 22, 22, 32)        0
_________________________________________________________________
batch_normalization_7 (Batch (None, 22, 22, 32)        128
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 20, 20, 64)        18496
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 10, 10, 64)        0
_________________________________________________________________
batch_normalization_8 (Batch (None, 10, 10, 64)        256
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               262272
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 258
=================================================================
Total params: 357,698
Trainable params: 357,506
Non-trainable params: 192
_________________________________________________________________
```

```
Train on 18966 samples, validate on 4742 samples
Epoch 1/11
18966/18966 [==============================] - 35s 2ms/sample - loss: 0.6846 -
accuracy: 0.5793 - val_loss: 0.6621 - val_accuracy: 0.5994
Epoch 2/11
18966/18966 [==============================] - 35s 2ms/sample - loss: 0.6483 -
accuracy: 0.6211 - val_loss: 0.6400 - val_accuracy: 0.6391
Epoch 3/11
18966/18966 [==============================] - 34s 2ms/sample - loss: 0.6282 -
accuracy: 0.6431 - val_loss: 0.6167 - val_accuracy: 0.6597
Epoch 4/11
18966/18966 [==============================] - 35s 2ms/sample - loss: 0.6050 -
accuracy: 0.6676 - val_loss: 0.5825 - val_accuracy: 0.6819
Epoch 5/11
18966/18966 [==============================] - 34s 2ms/sample - loss: 0.5834 -
accuracy: 0.6868 - val_loss: 0.5871 - val_accuracy: 0.6833
Epoch 6/11
18966/18966 [==============================] - 34s 2ms/sample - loss: 0.5689 -
accuracy: 0.7022 - val_loss: 0.5597 - val_accuracy: 0.7068
Epoch 7/11
18966/18966 [==============================] - 34s 2ms/sample - loss: 0.5448 -
accuracy: 0.7223 - val_loss: 0.5550 - val_accuracy: 0.7119
Epoch 8/11
18966/18966 [==============================] - 35s 2ms/sample - loss: 0.5248 -
accuracy: 0.7351 - val_loss: 0.5560 - val_accuracy: 0.7077
Epoch 9/11
18966/18966 [==============================] - 36s 2ms/sample - loss: 0.5046 -
accuracy: 0.7517 - val_loss: 0.5815 - val_accuracy: 0.6718
Epoch 10/11
18966/18966 [==============================] - 35s 2ms/sample - loss: 0.4849 -
accuracy: 0.7639 - val_loss: 0.5302 - val_accuracy: 0.7246
Epoch 11/11
18966/18966 [==============================] - 35s 2ms/sample - loss: 0.4702 -
accuracy: 0.7743 - val_loss: 0.5856 - val_accuracy: 0.6911
```

The model produced has training accuracy of 77% and validation accuracy of 70% and there is less gap in between training loss and validation loss, Hence this can be considered as out optimal model

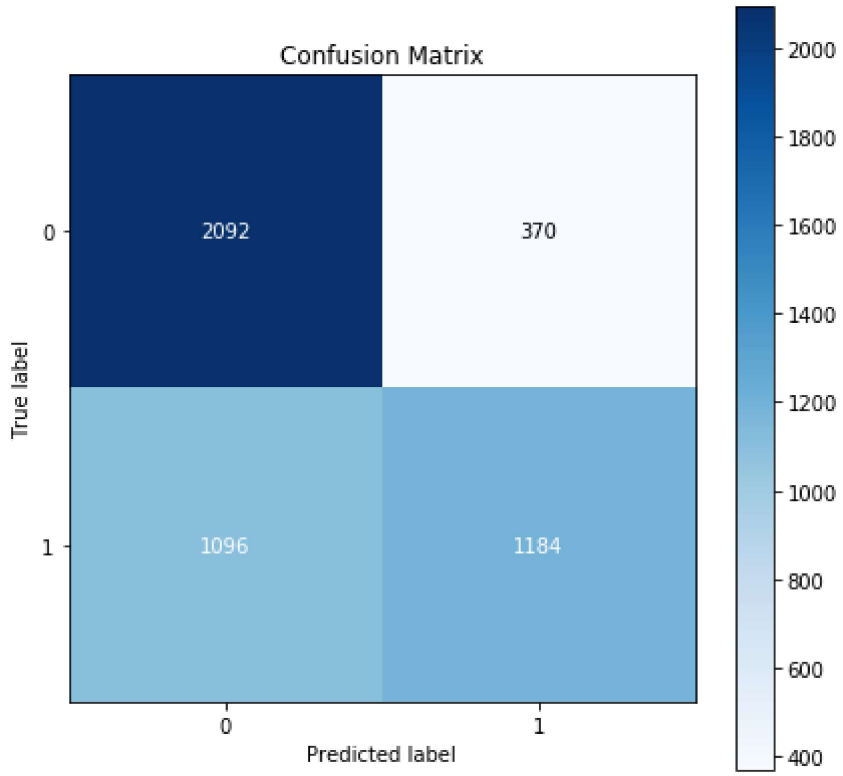
Plotting the Train Vs Validation Accuracy and Loss Graphs

In [19]:
```
fig = plt.figure(figsize=(12, 6))
ax = fig.add_subplot(1, 2, 1)
ax.plot(historyg2.history["loss"], 'r-x', label="Train Loss")
ax.plot(historyg2.history["val_loss"], 'b-x', label="Validation Loss")
ax.legend()
ax.set_title('Cross Entropy Loss')
ax.grid(True)

ax = fig.add_subplot(1, 2, 2)
ax.plot(historyg2.history["accuracy"], 'r-x', label="Train Accuracy")
ax.plot(historyg2.history["val_accuracy"], 'b-x', label="Validation Accuracy")
ax.legend()
ax.set_title('Accuracy')
ax.grid(True)
```



In [20]:
```
import scikitplot
y_valid = modelg2.predict_classes(X_test)
scikitplot.metrics.plot_confusion_matrix(np.argmax(y_test, axis=1), y_valid, fig

print(f'total wrong validation predictions: {np.sum(np.argmax(y_test, axis=1) !=
print(classification_report(np.argmax(y_test, axis=1), y_valid))
```

```
total wrong validation predictions: 1466


              precision    recall  f1-score   support

           0       0.66      0.85      0.74      2462
           1       0.76      0.52      0.62      2280

    accuracy                           0.69      4742
   macro avg       0.71      0.68      0.68      4742
weighted avg       0.71      0.69      0.68      4742
```



In [25]:
```
modelg2.save("C:/Users/nihal/Desktop/DL_2")
```

```
INFO:tensorflow:Assets written to: C:/Users/nihal/Desktop/DL_2\assets
```

Issues / Improvements

1. Lack of effective resources
2. Having only gray scale images for emotion detection.
3. Use cross-validaiton

References

- Dehghan, Afshin & Ortiz, Enrique & Shu, Guang & Masood, Syed. (2017). DAGER: Deep Age, Gender and Emotion Recognition Using Convolutional Neural Network.
- Özdemir, Mehmet & Elagoz, Berkay & Alaybeyoglu, Aysegul & Sadighzadeh, Reza & Akan, Aydin. (2019). Real Time Emotion Recognition from Facial Expressions Using CNN Architecture. 10.1109/TIPTEKNO.2019.8895215.
- Bargal, Sarah & Barsoum, Emad & Ferrer, Cristian & Zhang, Cha. (2016). Emotion recognition in the wild from videos using images. 433-436. 10.1145/2993148.2997627.
- Slides and notebooks uploaded in canvas.
- Levi, Gil & Hassncer, Tal. (2015). Age and gender classification using convolutional neural networks. 34-42. 10.1109/CVPRW.2015.7301352.
- Tal Hassner, Shai Harel, Eran Paz, Roee Enbar. Effective Face Frontalization in Unconstrained Images.
- https://riptutorial.com/keras/example/32608/transfer-learning-using-keras-and-vgg (https://riptutorial.com/keras/example/32608/transfer-learning-using-keras-and-vgg)
- https://towardsdatascience.com/predict-age-and-gender-using-convolutional-neural-network-and-opencv-fd90390e3ce6 (https://towardsdatascience.com/predict-age-and-gender-using-convolutional-neural-network-and-opencv-fd90390e3ce6)
- https://www.learnopencv.com/age-gender-classification-using-opencv-deep-learning-c-python/ (https://www.learnopencv.com/age-gender-classification-using-opencv-deep-learning-c-python/)
- https://www.researchgate.net/publication/225173459_Classification_of_Face_Images_for_Gend (https://www.researchgate.net/publication/225173459_Classification_of_Face_Images_for_Gen
- https://www.geeksforgeeks.org/image-classifier-using-cnn/ (https://www.geeksforgeeks.org/image-classifier-using-cnn/)

Credits

- https://www.kaggle.com/gauravsharma99/facial-emotion-recognition (https://www.kaggle.com/gauravsharma99/facial-emotion-recognition)