In [48]:
```
import numpy as np
import pandas as pd
import h5py
import random
import matplotlib.pyplot as plt
import seaborn as sns
from keras.callbacks import EarlyStopping,ModelCheckpoint
from keras.models import Sequential
from keras.layers import *
from keras.losses import binary_crossentropy
from sklearn.metrics import confusion_matrix,recall_score,precision_score,f1_score
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator
```

In [40]:
```
f_electron=h5py.File("ML4SCI_GSoC-main/SingleElectronPt50_IMGCROPS_n249k_RHv1.hdf5","r")
f_photon=h5py.File("ML4SCI_GSoC-main/SinglePhotonPt50_IMGCROPS_n249k_RHv1.hdf5","r")
```

- Splitting the dataset into training, validation and testing and then combining.
- Taking 60% as training, 20% validation and 20% as testing data.

In [42]:
```
train_split=int((f_electron["X"].shape[0])*0.6)
val_split=int((f_electron["X"].shape[0])*0.2)
```

In [43]:
```
X_train_photon=f_photon["X"][:train_split]
X_val_photon=f_photon["X"][train_split:train_split+val_split]
X_test_photon=f_photon["X"][train_split+val_split:]
Y_train_photon=f_photon["y"][:train_split]
Y_val_photon=f_photon["y"][train_split:train_split+val_split]
Y_test_photon=f_photon["y"][train_split+val_split:]

X_train_electron=f_electron["X"][:train_split]
X_val_electron=f_electron["X"][train_split:train_split+val_split]
X_test_electron=f_electron["X"][train_split+val_split:]
Y_train_electron=f_electron["y"][:train_split]
Y_val_electron=f_electron["y"][train_split:train_split+val_split]
Y_test_electron=f_electron["y"][train_split+val_split:]
```

In [44]:
```
combine_X_train=np.concatenate((X_train_photon,X_train_electron),axis=0)
combine_X_val=np.concatenate((X_val_photon,X_val_electron),axis=0)
combine_X_test=np.concatenate((X_test_photon,X_test_electron),axis=0)

combine_Y_train=np.concatenate((Y_train_photon,Y_train_electron),axis=0)
combine_Y_val=np.concatenate((Y_val_photon,Y_val_electron),axis=0)
combine_Y_test=np.concatenate((Y_test_photon,Y_test_electron),axis=0)
```

In [ ]:
```
random.shuffle(combine_X_train)  # shuffling of data so that we get random data.
random.shuffle(combine_X_val)
```

In [ ]:
```
np.save("TRAIN_DATA",combine_X_train)
np.save("VAL_DATA",combine_X_val)
```

Model Architecture

In [ ]:
```
batch_size=2500
```

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

model.add(Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=(batch_size,32,32,2)))
model.add(Conv2D(64,kernel_size=(3,3),activation='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(128,(3,3),activation='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(GlobalAveragePooling2D())
model.add(Dense(64,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1,activation='sigmoid'))
```

Model Compilation

In [12]:
```
model.compile(loss=binary_crossentropy,optimizer='adam',metrics=['accuracy'])
```

Creating Data Generators

In [35]:
```
def data_generator(path,batch=5000):
    gen=ImageDataGenerator()
    generator=gen.flow_from_directory(path,batch_size=batch,target_size=(32,32),class_mode=
"binary")
    return generator
```

In [ ]:
```
train_generator=data_generator("TRAIN_DATA",batch=batch_size)
val_generator=data_generator("VAL_DATA",batch=batch_size)
```

- Due To Limited Disk I am Running the dataset for only ten epoch.

Declaring Callbacks

In [ ]:
```
EarlyStop=EarlyStopping(monitor="val_acc",patience=1)
Modelcheck=ModelCheckpoint("best_model.h5",monitor="val_loss",verbose=0,save_best_only=True,
save_weights_only=False)
```

In [ ]:
```
hist=model.fit_generator(tarin_generator,steps_per_epoch=60,validation_data=val_generator,
                         validation_steps=20,callbacks=[EarlyStop,Modelcheck],epochs=10)
```

Visualization of Accuracy and Loss Epoch Wise

In [ ]:
```
val_loss=hist.history['val_loss']
train_loss=hist.history['loss']
train_acc=hist.history['accuracy']
val_acc=hist.history['val_accuracy']
```

In [ ]:
```
# Generating Plot for the loss.
plt.figure(figsize=(15,8))
plt.title("Loss vs Epoch",fontsize=16)
plt.plot(val_loss,c='red',label="Validation Loss")
plt.plot(train_loss,c="navy",label="Training Loss")
plt.xlabel("Epoch",fontsize=14)
plt.ylabel("Loss",fontsize=14)
plt.style.use("seaborn")
plt.show()
```

In [ ]:
```
# Generating Plot for accuracy.
plt.figure(figsize=(15,8))
plt.title("Accuracy vs Epoch",fontsize=16)
plt.plot(val_acc,c='red',label="Validation Accuracy")
plt.plot(train_acc,c="navy",label="Training Accuracy")
plt.xlabel("Epoch",fontsize=14)
plt.ylabel("Accuracy",fontsize=14)
plt.style.use("seaborn")
plt.show()
```

Making Predictions

In [ ]:
```
predictions=np.argmax(model.predict(combine_X_test),axis=1)
```

In [ ]:
```
conf_matrix=confusion_matrix(combine_Y_test,predictions)
```

In [ ]:
```
# Generating Confusion Matrix
plt.figure(figsize=(15,8))
sns.heatmap(conf_matrix,annot=True)
plt.show()
```

Examining How good Was The Prediction

In [ ]:
```
accuracy=(np.diag(conf_matrix).sum()/np.sum(conf_matrix))*100   # Accuracy of Model
recall=recall_score(combine_Y_test,predictions)*100   # Recall Score of Model
precision=precision_score(combine_Y_test,predictions)*100  # Precision Score of Model
fscore=f1_score(combine_Y_test,predictions)*100   # F1-Score of Model
```

In [53]:
```
metrics_df=pd.DataFrame([accuracy,recall,precision,fscore],index=['Accuracy','Recall Score',
'Precision Score','F-Score'],columns=['Model Evaluation Metrics (%)'])
```

In [ ]:
```
metrics_df
```

In [ ]:

In [ ]: