Common Task 1: Electron/photon classification (TensorFlow)

In [5]:
```
# I need to downscale tensorflow to 2.9.1 as i was using TPU on kaggle
from IPython.display import clear_output
!pip install -q /lib/wheels/tensorflow-2.9.1-cp38-cp38-linux_x86_64.whl
!pip install scikit-learn
clear_output()
```

In [ ]:
```
# Import necessary packages
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import h5py
import urllib
import math
```

In [ ]:
```
# initialize TPU
print('TensorFlow Version:', tf.__version__)

try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver.connect(tpu="local")
    strategy = tf.distribute.TPUStrategy(tpu)

except Exception as e:
    print(e)
    strategy = tf.distribute.get_strategy()
```

In [8]:
```
# Downloading Dataset
def download_file(url, filename):
    urllib.request.urlretrieve(url, filename)

photon_url = 'https://cernbox.cern.ch/remote.php/dav/public-files/AtBT8y4MiQYFcgc/SinglePhotonPt50_IMGCROPS_n249k_RHv1.hd
electron_url = 'https://cernbox.cern.ch/remote.php/dav/public-files/FbXw3V4XNyYB3oA/SingleElectronPt50_IMGCROPS_n249k_RHv

download_file(photon_url, 'photon.hdf5')
download_file(electron_url, 'electron.hdf5')
```

Load the Data

In [9]:

```
# Define the filenames and batch size
electrons_filename = 'electron.hdf5'
photons_filename = 'photon.hdf5'
batch_size = 64
num_features = 32

# Define a generator function to load and process data in batches
def batch_generator(electrons_filename, photons_filename, batch_size):
    # Open the HDF5 files and get the number of samples
    electrons_file = h5py.File(electrons_filename, 'r')
    photons_file = h5py.File(photons_filename, 'r')
    num_electrons = electrons_file['X'].shape[0]
    num_photons = photons_file['X'].shape[0]

    # Calculate the number of batches per epoch
    num_batches = max(num_electrons, num_photons) // batch_size

    # Loop over the data and yield batches
    for i in range(num_batches):
        # Load a batch of electrons and photons
        electrons_x = electrons_file['X'][i*batch_size:(i+1)*batch_size]
        electrons_y = electrons_file['y'][i*batch_size:(i+1)*batch_size]

        photons_x = photons_file['X'][i*batch_size:(i+1)*batch_size]
        photons_y = photons_file['y'][i*batch_size:(i+1)*batch_size]

        # Combine the data
        batch_x = np.concatenate([electrons_x, photons_x])
        batch_y = np.concatenate([electrons_y, photons_y])

        # expand dims of batch_y
        batch_y = np.expand_dims(batch_y, axis=1)

        # shuffle it
        perm = np.random.permutation(len(batch_x))
        batch_x = batch_x[perm]
        batch_y = batch_y[perm]

        # Convert the data to TensorFlow tensors and yield it
        yield tf.convert_to_tensor(batch_x, dtype=tf.float32), tf.convert_to_tensor(batch_y, dtype=tf.int32)
```

```python
    # Close the HDF5 files
    electrons_file.close()
    photons_file.close()

# Create a TensorFlow Dataset from the generator function
batched_dataset = tf.data.Dataset.from_generator(
    lambda: batch_generator(electrons_filename, photons_filename, batch_size//2),
    output_types=(tf.float32, tf.int32),
    output_shapes=((batch_size, num_features, num_features, 2), (batch_size, 1))
).repeat()
```

In [10]:

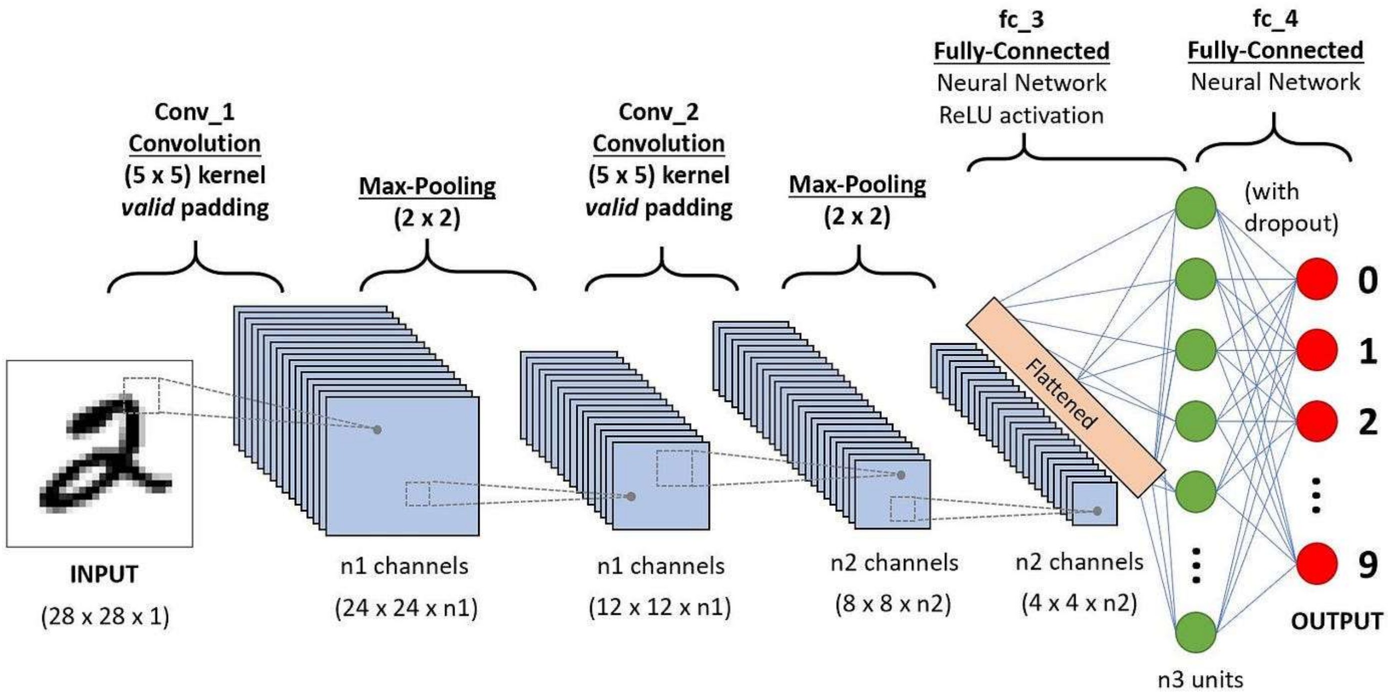
```python
train_frac = 0.032 # using only 0.032% of the entire dataset

# calculating dataset size
dataset_size = math.ceil((h5py.File(electrons_filename, 'r')['X'].shape[0]*2)/batch_size)

# Define the size of the training and testing datasets
train_size = math.ceil(dataset_size * train_frac)
test_size = math.floor(dataset_size * (1 - train_frac))

train_dataset = batched_dataset.take(train_size)
test_dataset = batched_dataset.skip(train_size).take(test_size)
```

The Model Architecture



A convolutional neural network (CNN) is a type of deep learning algorithm that is particularly well-suited for image recognition and processing tasks. It is made up of multiple layers, including convolutional layers, pooling layers, and fully connected layers. Convolutional layers apply filters to the input data and produce feature maps that capture the spatial information of the data. Pooling layers reduce the dimensionality of the feature maps and introduce some invariance to translation, rotation, and scaling. Fully connected layers perform classification based on the extracted features.

Some examples of CNN architectures are LeNet, AlexNet, VGG, ResNet, and Inception.

I will also like to add that I have created a detailed youtube video on how CNN works: https://youtu.be/H1ZC9COwtMs

In [11]:

```
# creating model

with strategy.scope():

    model = tf.keras.Sequential([
        tf.keras.layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu'),
        tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
        tf.keras.layers.MaxPooling2D((2,2)),
        tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
```

```python
        tf.keras.layers.MaxPooling2D((2,2)),
        tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
        tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
        tf.keras.layers.MaxPooling2D((2,2)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(units=64, activation='relu'),
        tf.keras.layers.Dense(units=1, activation='sigmoid')
    ])

    model.compile(
        optimizer=tf.keras.optimizers.Adam(),
        loss=tf.keras.losses.BinaryCrossentropy(),
        metrics=['accuracy', tf.keras.metrics.AUC(curve='ROC')]
    )
```

In [13]:

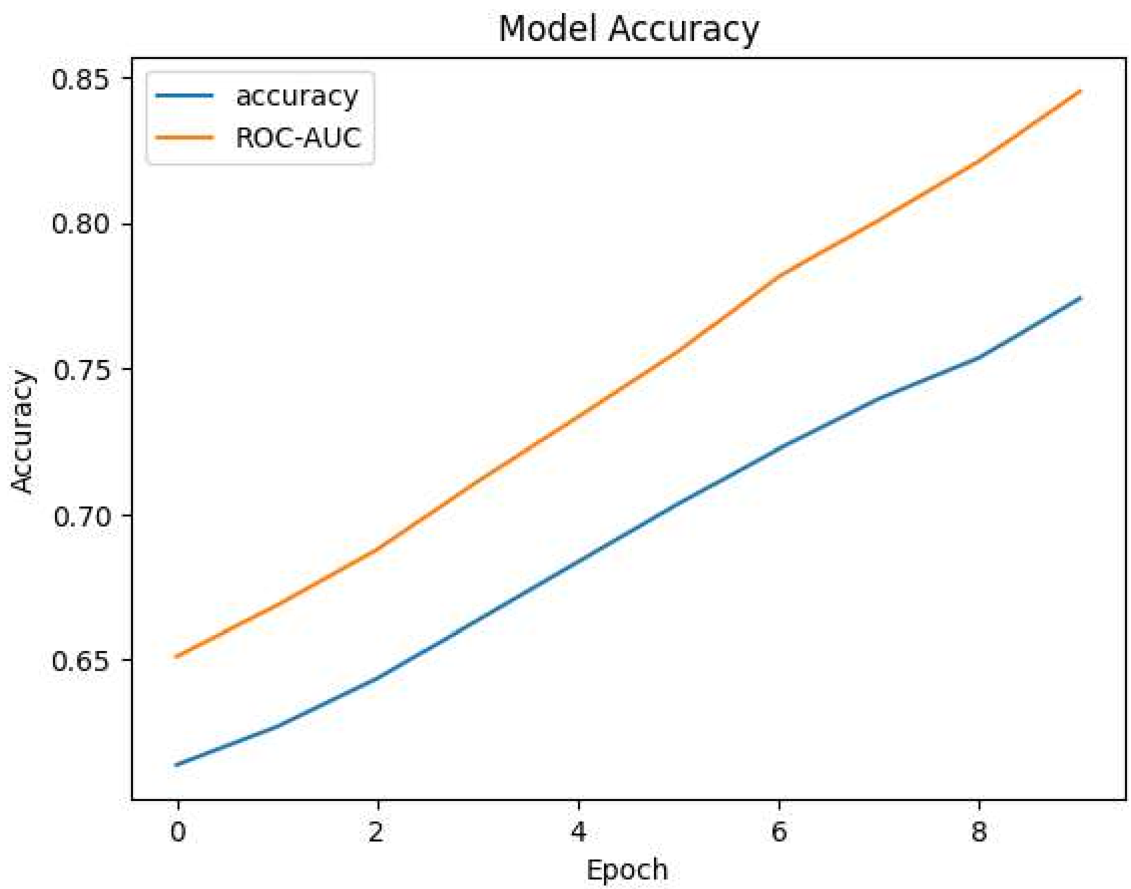
```python
history = model.fit(
    train_dataset,
    epochs=10
)
```

```
Epoch 1/10
250/250 [==============================] - 79s 317ms/step - loss: 0.6541 - accuracy: 0.6138 - auc: 0.6511
Epoch 2/10
250/250 [==============================] - 79s 314ms/step - loss: 0.6446 - accuracy: 0.6269 - auc: 0.6686
Epoch 3/10
250/250 [==============================] - 79s 316ms/step - loss: 0.6326 - accuracy: 0.6436 - auc: 0.6878
Epoch 4/10
250/250 [==============================] - 79s 316ms/step - loss: 0.6168 - accuracy: 0.6635 - auc: 0.7113
Epoch 5/10
250/250 [==============================] - 79s 316ms/step - loss: 0.6000 - accuracy: 0.6836 - auc: 0.7333
Epoch 6/10
250/250 [==============================] - 79s 315ms/step - loss: 0.5811 - accuracy: 0.7036 - auc: 0.7559
Epoch 7/10
250/250 [==============================] - 79s 317ms/step - loss: 0.5572 - accuracy: 0.7224 - auc: 0.7817
Epoch 8/10
250/250 [==============================] - 79s 314ms/step - loss: 0.5370 - accuracy: 0.7396 - auc: 0.8009
Epoch 9/10
250/250 [==============================] - 79s 317ms/step - loss: 0.5141 - accuracy: 0.7538 - auc: 0.8213
Epoch 10/10
250/250 [==============================] - 79s 317ms/step - loss: 0.4837 - accuracy: 0.7742 - auc: 0.8453
```

In [15]:

```python
plt.title('Model Accuracy')
plt.plot(history.history['accuracy'], label='accuracy')
```

```python
plt.plot(history.history['auc'], label='ROC-AUC')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```



ROC AUC SCORE: 0.8453