```
from tensorflow.data import Dataset

"""
x_train contains all data, and count_training represents the distribution
test/evaluation, e.g. 80/20, 70/30, etc.
"""
train_ds = Dataset.from_tensor_slices((x_train[:count_training], y_train[:count_training]))
validation_ds = Dataset.from_tensor_slices((x_train[count_training:], y_train[count_training:]))
test_ds = Dataset.from_tensor_slices((x_test, y_test))
```

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D, MaxPool2D, BatchNormalization, Flatten, Dropout, Dens
from tensorflow.keras.regularizers import l2
from tensorflow.keras.activations import relu, sigmoid
from tensorflow.keras.initializers import GlorotNormal

#this configuration uses backend.set_image_data_format('channels_first')

"""
This function creates a model composed by two convolutional + max pooling layers.
After, to standardize the input a batch normalization is applied.
This is put into a single one-dimensional layer using the Flatten layer.
Next, the dropout regularization technique (50%) is perfomed.
Finally, Dense layer output for binary classification
"""
def get_model_design(filters: list, input_shape: tuple) -> Sequential:
    model = Sequential([Conv2D(filters[0], (5, 5),  padding='same', kernel_regularizer=l2(0.001),
                        Conv2D(filters[1], (3, 3),  padding='same', kernel_regularizer=l2(0.001),
                        MaxPool2D(pool_size=(2, 2)),
                        BatchNormalization(),
                        Flatten(),
                        Dropout(0.5),
                        Dense(1, kernel_initializer=GlorotNormal(), activation=sigmoid )
                       ])
    return model

# for this example, we used 128 and 64 filters for the two first conv layers
# note the input size of 3 channel for an image size of 64x64 pixels
model = get_model_design([128, 64], (3, 64, 64))
model.summary()
```

```python
from tensorflow.keras import Model
from tensorflow.keras.layers import Input, Convolution2D, MaxPool2D, BatchNormalization, Flatten,
from tensorflow.keras.regularizers import l2
from tensorflow.keras.activations import relu, sigmoid
from tensorflow.keras.initializers import GlorotNormal

# this configuration uses backend.set_image_data_format('channels_first')

"""
This design creates the same network than before but using the layer by layer configuration.
Notice the Input layer and each layer description.
Function returns a model which require inputs and outputs (could be multiple of each one)
"""
def get_model_design(filters: list, input_shape: tuple) -> Model:
    input_layer = Input(shape=input_shape)

    conv1_layer = Convolution2D(filters[0], (5, 5), padding='same', kernel_regularizer=l2(0.001),
    conv2_layer = Convolution2D(filters[1], (3, 3), padding='same', kernel_regularizer=l2(0.001),
    maxpool1_layer = MaxPool2D(pool_size=(2, 2))(conv2_layer)
    norm1_layer = BatchNormalization()(maxpool1_layer)

    flat1_layer = Flatten()(norm1_layer)
    drop1_layer = Dropout(0.5)(flat1_layer)
    pred_layer = Dense(1, kernel_initializer=GlorotNormal(), activation=sigmoid)(drop1_layer)

    model = Model(inputs=input_layer, outputs=pred_layer)
    return model

# for this example, we used 128 and 64 filters for the two first conv layers
# note the input size of 3 channel for an image size of 64x64 pixels
model = get_model_design([128, 64], (3, 64, 64))
model.summary()
```

```python
from tensorflow.keras.utils import plot_model

plot_model(model, 'my-CNNmodel.png', show_shapes=True)
```

```python
from tensorflow.keras.losses import BinaryCrossentropy
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import TruePositives, FalsePositives, TrueNegatives, FalseNegatives
from tensorflow.keras.metrics import SpecificityAtSensitivity

"""
Definition of metrics commonly used on medical imaging classification, segmentation, and localiz
The metrics will appear on each iteration of the training process to monitor the progress of our
"""
METRICS = [
      TruePositives(name='tp'),
      FalsePositives(name='fp'),
      TrueNegatives(name='tn'),
      FalseNegatives(name='fn'),
      BinaryAccuracy(name='accuracy'),
      Precision(name='precision'),
      Recall(name='recall'),
      AUC(name='auc'),
      SpecificityAtSensitivity(sensitivity=0.8, name='sensitivity'),
]

"""
For example, the loss function is to determine is an image contains or not a lesion/disease usin
The optimizer is a first-order gradient-based optimization
"""
model.compile(loss=BinaryCrossentropy(),
              optimizer=Adam(lr=1e-3, beta_1=0.92, beta_2=0.999),
              metrics=METRICS)
```

```python
from tensorflow.keras.callbacks import EarlyStopping

"""
This callback will stop the training when there is no improvement in the validation accuracy acr
"""
early_callback = EarlyStopping(monitor='val_auc',
                               verbose=1,
                               patience=10,
                               mode='max',
                               restore_best_weights=True)
```

```
batch_size = 64

"""
Training the model for 60 epochs using our dataset.
The batch size (64) is the same for the validation data.
Only 1 callback was used, but could be more like TensorBoard, ModelCheckpoint, etc.
"""
history = model.fit(train_ds.batch(batch_size=batch_size),
                    epochs=60,
                    validation_data=validation_ds.batch(batch_size=batch_size),
                    callbacks=[early_callback])
model.save('model_base')
```

```
import matplotlib.pyplot as plt
from matplotlib import rcParams

rcParams['figure.figsize'] = (12, 10)
colors = plt.rcParams['axes.prop_cycle'].by_key()['color']

def plot_log_loss(history: History, title_label: str, n: int) -> ():
    # Use a log scale to show the wide range of values.
    plt.semilogy(history.epoch,  history.history['loss'],
                 color=colors[n], label='Train '+title_label)
    plt.semilogy(history.epoch,  history.history['val_loss'],
            color=colors[n], label='Val '+title_label,
            linestyle="--")
    plt.xlabel('Epoch')
    plt.ylabel('Loss')

    plt.legend()

plot_log_loss(history, "Model Base", 1)
```

```
def plot_metrics(history: History) -> ():
    metrics =  ['loss', 'precision', 'recall', 'auc', 'tp', 'sensitivity']
    for n, metric in enumerate(metrics):
        name = metric.replace("_"," ").capitalize()
        plt.subplot(3, 2, n+1)  # adjust according to metrics
        plt.plot(history.epoch,  history.history[metric], color=colors[0], label='Train')
        plt.plot(history.epoch, history.history['val_'+metric],
                 color=colors[0], linestyle="--", label='Val')
        plt.xlabel('Epoch')
        plt.ylabel(name)
        # selecting the metric, the value of plt.ylim could be changed
    plt.legend()

plot_metrics(history)
```

```python
# Evaluate the model on the test data using `evaluate`
print("Evaluate on test data")
score_test = model.evaluate(test_ds.batch(batch_size))
for name, value in zip(model.metrics_names, score_test):
    print(name, ': ', value)
```

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

# notice the threshold
def plot_cm(labels: numpy.ndarray, predictions: numpy.ndarray, p: float=0.5) -> ():
    cm = confusion_matrix(labels, predictions > p)
    # you can normalize the confusion matrix

    plt.figure(figsize=(5,5))
    sns.heatmap(cm, annot=True, fmt="d")
    plt.title('Confusion matrix @{:.2f}'.format(p))
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')

    print('Lesions Detected (True Negatives): ', cm[0][0])
    print('Lesions Incorrectly Detected (False Positives): ', cm[0][1])
    print('No-Lesions Missed (False Negatives): ', cm[1][0])
    print('No-Lesions Detected (True Positives): ', cm[1][1])
    print('Total Lesions: ', np.sum(cm[1]))

plot_cm(y_test, y_test_pred)
```

```python
precision = Precision()
precision.update_state(y_train, y_train_pred)
precision.result().numpy()
```

```python
from sklearn.metrics import roc_auc_score, roc_curve

def plot_roc(name: str, labels: numpy.ndarray, predictions: numpy.ndarray, **kwargs) -> ():
  fp, tp, _ = roc_curve(labels, predictions)
  auc_roc = roc_auc_score(labels, predictions)
  plt.plot(100*fp, 100*tp, label=name + " (" + str(round(auc_roc, 3)) + ")",
           linewidth=2, **kwargs)
  plt.xlabel('False positives [%]')
  plt.ylabel('True positives [%]')
  plt.title('ROC curve')
  plt.grid(True)
  plt.legend(loc='best')
  ax = plt.gca()
  ax.set_aspect('equal')

plot_roc("Train Base", y_train, y_train_pred, color=colors[0])
plot_roc("Test Base", y_test, y_test_pred, color=colors[0], linestyle='--')
plt.legend(loc='lower right')
```