In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import cv2
import gc

from PIL import Image
train_on_gpu = True

from sklearn.utils import resample
from sklearn.utils import shuffle
from sklearn.metrics import roc_auc_score

import torchvision.transforms as transforms
from torch.utils.data.sampler import SubsetRandomSampler
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import TensorDataset, DataLoader, Dataset
import torchvision
import torch.optim as optim
import torchvision.models as models

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dens
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping# This Python 3 env
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWar
ning: A NumPy version >=1.16.5 and <1.23.0 is required for this versio
n of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversio
n}"
```

In [3]:

```
# helper method for clearing GPU memory
def clear_memory():
    gc.collect()
    torch.cuda.empty_cache()
```

In [4]:
```python
df_train = pd.read_csv("../input/histopathologic-cancer-detection/train_
df_sample_sub = pd.read_csv("../input/histopathologic-cancer-detection/
df_train.head()
```

Out[4]:

	id	label
0	f38a6374c348f90b587e046aac6079959adf3835	0
1	c18f2d887b7ae4f6742ee445113fa1aef383ed77	1
2	755db6279dae599ebb4d39a9123cce439965282d	0
3	bc3f0c64fb968ff4a8bd33af6971ecae77c75e08	0
4	068aba587a4950175d04c680d38943fd488d6a9d	0

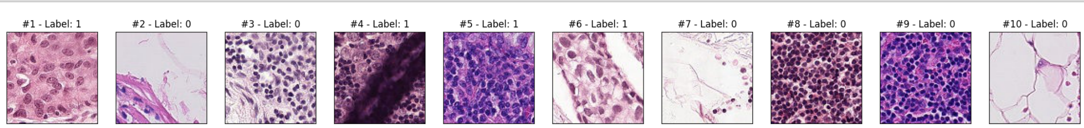
In [5]:
```python
folder_train = "../input/histopathologic-cancer-detection/train/"
folder_test = "../input/histopathologic-cancer-detection/test/"

print("Number of training images: {}".format(len(os.listdir(folder_trai
print("Number of test images: {}".format(len(os.listdir(folder_test)))
```

```
Number of training images: 220025
Number of test images: 57458
```

In [6]:
```python
# Load the images
img_train = os.listdir(folder_train)
img_test = os.listdir(folder_test)
```

In [7]:
```python
# print the first 10 images
fig = plt.figure(figsize=(25, 4))
for i in range(10):
    ax = fig.add_subplot(1, 10, i + 1, xticks=[], yticks=[])
    im = Image.open(folder_train + img_train[i])
    plt.imshow(im)
    label = df_train.loc[df_train['id'] == img_train[i].split('.')[0],
    ax.set_title(f'#{i+1} - Label: {label}')
```



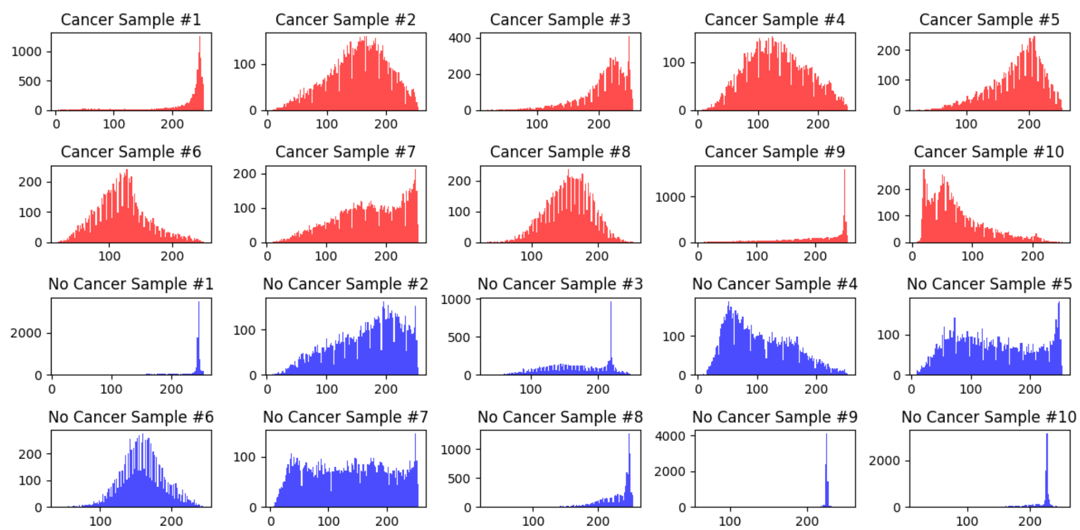
In [8]:

```python
# sample first 10 images for both labels
cancer_samples = df_train[df_train['label'] == 1].head(10)
no_cancer_samples = df_train[df_train['label'] == 0].head(10)

# plot histograms of image pixel values for cancer and no cancer images
plt.figure(figsize=(12, 6))
for i in range(10):
    plt.subplot(4, 5, i + 1)
    cancer_img = cv2.imread(folder_train + cancer_samples.iloc[i]['id']
    plt.hist(cancer_img.ravel(), bins=128, color='red', alpha=0.7)
    plt.title(f'Cancer Sample #{i+1}')

    plt.subplot(4, 5, i + 11)
    no_cancer_img = cv2.imread(folder_train + no_cancer_samples.iloc[i]
    plt.hist(no_cancer_img.ravel(), bins=128, color='blue', alpha=0.7)
    plt.title(f'No Cancer Sample #{i+1}')

plt.tight_layout()
plt.show()
```

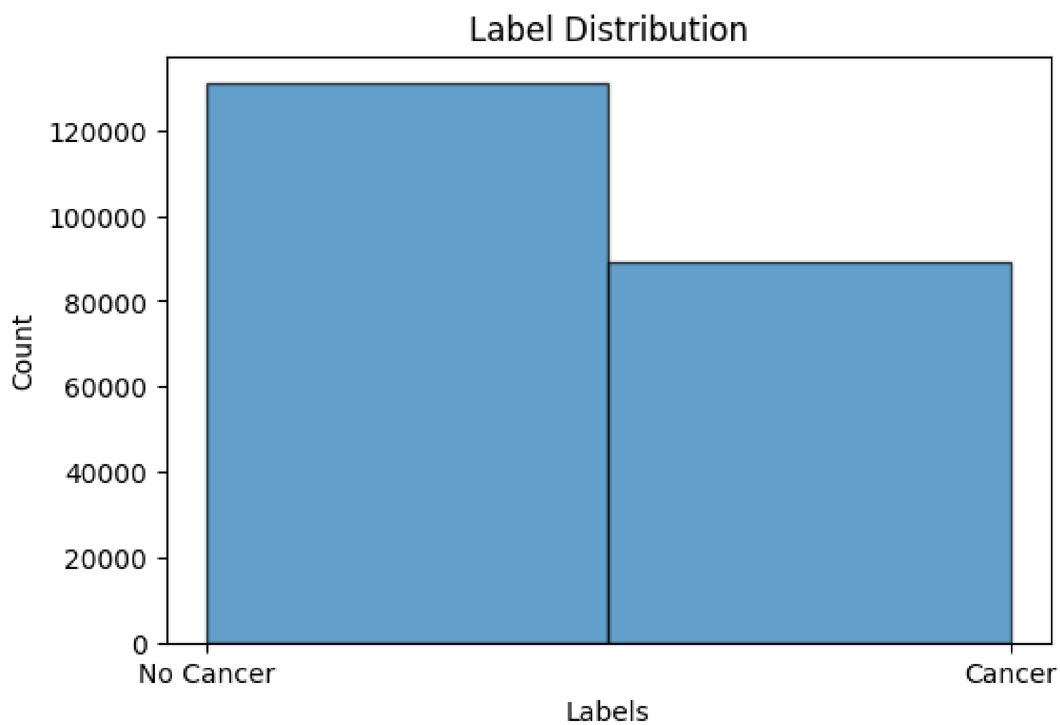


In [11]:

```python
# plot histogram of label distribution
def plot_label_dist(df):
    plt.figure(figsize=(6, 4))
    plt.hist(df['label'], bins=2, edgecolor='black', alpha=0.7)
    plt.xticks(np.arange(2), ['No Cancer', 'Cancer'])
    plt.xlabel('Labels')
    plt.ylabel('Count')
    plt.title('Label Distribution')
    plt.show()
```

In [12]:

```python
plot_label_dist(df_train)
```



In [13]:

```python
# calculate the imbalance ratios
def calc_imbalance(df_train):

    df_train_cancer = df_train[df_train['label'] == 1]
    df_train_no_cancer = df_train[df_train['label'] == 0]

    cancer = len(df_train_cancer)
    no_cancer = len(df_train_no_cancer)

    imbalance_ratio = no_cancer / cancer
    cancer_ratio = cancer / (cancer + no_cancer)

    print("Imbalance ratio:", round(imbalance_ratio, 3))
    print("Ratio of cancer:", round(cancer_ratio, 3))
```

In [14]:

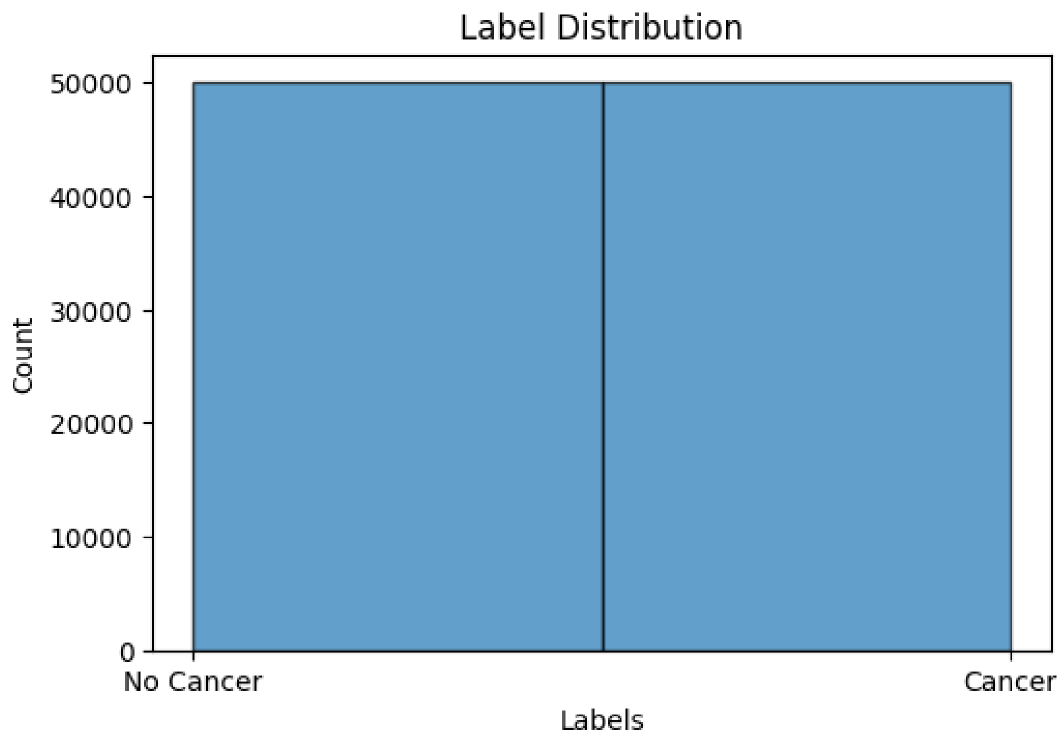
```python
calc_imbalance(df_train)
```

```
Imbalance ratio: 1.469
Ratio of cancer: 0.405
```

In [15]:
```
# sample positive and negative images
sample_size = 50000
df_train_neg = df_train[df_train['label'] == 0].sample(sample_size, ran
df_train_pos = df_train[df_train['label'] == 1].sample(sample_size, ran

# create a new shuffeled training dataset
df_train_sample = shuffle(pd.concat([df_train_pos, df_train_neg], axis=
```

In [16]:
```
plot_label_dist(df_train_sample)
```



In [17]:

```
# wrapper class for PyTorch dataset
class PyTorchData(Dataset):

    # set the necessary super class properties
    def __init__(self, df, folder = './', transform=None):
        super().__init__()
        self.df = df.values
        self.data_dir = folder
        self.transform = transform

    # returns the length of the dataset
    def __len__(self):
        return len(self.df)

    # returns the image with the given index and applies a transformati
    def __getitem__(self, index):
        img_name,label = self.df[index]
        img_path = os.path.join(self.data_dir, img_name+'.tif')
        image = cv2.imread(img_path)
        if self.transform is not None:
            image = self.transform(image)
        return image, label
```

In [18]:

```
transform_train = transforms.Compose([
    transforms.ToPILImage(),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.RandomRotation(20),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5],std=[0.5, 0.5, 0.5])
])

train_torch = PyTorchData(df_train_sample, folder_train, transform_trai
```

In [19]:

```
batch_size = 128

# set training and validation indices
indices = list(range(len(train_torch)))
split = int(np.floor(0.15 * len(train_torch)))
train_idx, valid_idx = indices[split:], indices[:split]

# random samplers
train_sampler = SubsetRandomSampler(train_idx)
valid_sampler = SubsetRandomSampler(valid_idx)

# prepare data loaders
train_loader = DataLoader(train_torch, batch_size=batch_size, sampler=t
valid_loader = DataLoader(train_torch, batch_size=batch_size, sampler=v
```

In [20]:
```
transform_test = transforms.Compose([
    transforms.ToPILImage(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5],std=[0.5, 0.5, 0.5])
])

test_torch = PyTorchData(df_sample_sub, folder_test, transform_test)
test_loader = DataLoader(test_torch, batch_size=batch_size, shuffle=Fal
```

In [21]:
```
clear_memory()
```

In [22]:
```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
print("Device: ", device)
```

```
Device:  cuda:0
```

In [23]:

```python
class CNN(nn.Module):
    def __init__(self):
        super(CNN,self).__init__()

        self.conv1 = nn.Sequential(
                        nn.Conv2d(3, 32, 3, stride=1, padding=1),
                        nn.BatchNorm2d(32),
                        nn.ReLU(inplace=True),
                        nn.MaxPool2d(2,2))

        self.conv2 = nn.Sequential(
                        nn.Conv2d(32, 64, 3, stride=1, padding=1),
                        nn.BatchNorm2d(64),
                        nn.ReLU(inplace=True),
                        nn.MaxPool2d(2,2))

        self.conv3 = nn.Sequential(
                        nn.Conv2d(64, 128, 3, stride=1, padding=1),
                        nn.BatchNorm2d(128),
                        nn.ReLU(inplace=True),
                        nn.MaxPool2d(2,2))

        self.conv4 = nn.Sequential(
                        nn.Conv2d(128, 256, 3, stride=1, padding=1),
                        nn.BatchNorm2d(256),
                        nn.ReLU(inplace=True),
                        nn.MaxPool2d(2,2))

        self.conv5 = nn.Sequential(
                        nn.Conv2d(256, 512, 3, stride=1, padding=1),
                        nn.BatchNorm2d(512),
                        nn.ReLU(inplace=True),
                        nn.MaxPool2d(2,2))


        self.fc=nn.Sequential(
                nn.Linear(512*3*3, 256),
                nn.ReLU(inplace=True),
                nn.BatchNorm1d(256),
                nn.Dropout(0.4),
                nn.Linear(256, 1))

    def forward(self,x):
        x=self.conv1(x)
        x=self.conv2(x)
        x=self.conv3(x)
        x=self.conv4(x)
        x=self.conv5(x)
        x=x.view(x.shape[0],-1)
        x=self.fc(x)
        return x
```

In [24]:

```python
# create CNN
model = CNN().to(device)
print(model)
```

```
CNN(
  (conv1): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, cei
l_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_r
unning_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, cei
l_mode=False)
  )
  (conv3): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, cei
l_mode=False)
  )
  (conv4): Sequential(
    (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, cei
l_mode=False)
  )
  (conv5): Sequential(
    (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=
(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, cei
l_mode=False)
  )
  (fc): Sequential(
    (0): Linear(in_features=4608, out_features=256, bias=True)
    (1): ReLU(inplace=True)
    (2): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
    (3): Dropout(p=0.4, inplace=False)
    (4): Linear(in_features=256, out_features=1, bias=True)
  )
)
```

In [26]:

```
# hyperparameters to tune
learning_rates = [0.001, 0.0005, 0.0002]
```

In [ ]:

```python
# save final results
results = []
best_model_idx = None
best_auc = 0.0
best_model_cnn = None

# iterate all values for the hyperparameter
for idx, lr in enumerate(learning_rates):

    # use GPU if available
    model = CNN().to(device)

    # define optimizer and loss function
    optimizer = optim.Adam(model.parameters(), lr=lr)
    criterion = nn.BCEWithLogitsLoss()

    # save results over epochs
    train_losses = []
    valid_losses = []
    valid_aucs = []
    n_epochs = 10

    # iterate all epochs
    for epoch in range(1, n_epochs+1):

        valid_aucs_epoch = []
        model.train()
        train_loss = 0.0

        # process the training images
        for data, target in train_loader:
            optimizer.zero_grad()
            data, target = data.to(device), target.to(device)
            output = model(data)
            loss = criterion(output.view(-1), target.float())
            loss.backward()
            optimizer.step()
            train_loss += loss.item() * data.size(0)

        model.eval()
        valid_loss = 0.0

        # process the validation images
        for data, target in valid_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            loss = criterion(output.view(-1), target.float())
            valid_loss += loss.item() * data.size(0)
            y_actual = target.data.cpu().numpy()
            y_pred = torch.sigmoid(output).detach().cpu().numpy()
            valid_aucs_epoch.append(roc_auc_score(y_actual, y_pred))

        # determine values for current epoch
        train_loss /= len(train_loader.sampler)
        valid_loss /= len(valid_loader.sampler)
        valid_auc = np.mean(valid_aucs_epoch)
```

```
        train_losses.append(train_loss)
        valid_losses.append(valid_loss)
        valid_aucs.append(valid_auc)

        print('Learning Rate: {:.6f} | Epoch: {} | Training Loss: {:.6f

    # save results for current value of hyperparameter
    results.append({'learning_rate': lr, 'train_losses': train_losses,

    # save best model
    avg_auc = np.mean(valid_aucs)
    if avg_auc > best_auc:
        best_auc = avg_auc
        best_model_idx = idx
        best_model_cnn = model

# print best hyperparameter
print("Best learning rate according to hyperparameter search: ", learni
```

In [ ]:

```
# plot training and validation loss over epochs for best model
def plot_losses(train_losses, valid_losses, title):
    plt.figure(figsize=(10, 5))
    plt.plot(train_losses, label="Train Loss")
    plt.plot(valid_losses, label="Valid Loss")
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title(title)
    plt.legend()
    plt.show()
```

In [ ]:

```
best_result = results[best_model_idx]
plot_losses(best_result['train_losses'], best_result['valid_losses'], '
```

In [ ]:

```
# plot validation auc over epochs for best model
def plot_aucs(aucs, title):
    plt.figure(figsize=(10, 5))
    plt.plot(aucs, label="Validation AUC")
    plt.xlabel('Epochs')
    plt.ylabel('Validation AUC')
    plt.title(title)
    plt.legend()
    plt.show()
```

In [ ]:

```
best_result = results[best_model_idx]
plot_aucs(best_result['valid_aucs'], 'Validation AUC over Epochs (best 
```

In [ ]:

```
clear_memory()
```

In [ ]:

```
class DenseNetModified(nn.Module):
    def __init__(self):
        super(DenseNetModified, self).__init__()

        # use a pretrained dense net architecture
        self.densenet = models.densenet121(pretrained=True)

        num_features = self.densenet.classifier.in_features
        self.densenet.classifier = nn.Sequential(
            nn.Linear(num_features, 512),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(512, 1)
        )

    def forward(self, x):
        return self.densenet(x)
```

In [ ]:

```python
# reuse best learning rate from first CNN
lr = learning_rates[best_model_idx]
model_dense = DenseNetModified().to(device)

# set optimizer and loss function
optimizer = optim.Adam(model_dense.parameters(), lr=lr)
criterion = nn.BCEWithLogitsLoss()

# store final results
train_losses_dense = []
valid_losses_dense = []
valid_aucs_dense = []
n_epochs = 10

# iterate all epochs
for epoch in range(1, n_epochs+1):

    valid_aucs_epoch = []
    model_dense.train()
    train_loss = 0.0

    # process training images
    for data, target in train_loader:
        optimizer.zero_grad()
        data, target = data.to(device), target.to(device)
        output = model_dense(data)
        loss = criterion(output.view(-1), target.float())
        loss.backward()
        optimizer.step()
        train_loss += loss.item() * data.size(0)

    model_dense.eval()
    valid_loss = 0.0

    # process validation images
    for data, target in valid_loader:
        data, target = data.to(device), target.to(device)
        output = model_dense(data)
        loss = criterion(output.view(-1), target.float())
        valid_loss += loss.item() * data.size(0)
        y_actual = target.data.cpu().numpy()
        y_pred = torch.sigmoid(output).detach().cpu().numpy()
        valid_aucs_epoch.append(roc_auc_score(y_actual, y_pred))

    # store final results
    train_loss /= len(train_loader.sampler)
    valid_loss /= len(valid_loader.sampler)
    valid_auc = np.mean(valid_aucs_epoch)

    train_losses_dense.append(train_loss)
    valid_losses_dense.append(valid_loss)
    valid_aucs_dense.append(valid_auc)
```

```
    print('Learning Rate: {:.6f} | Epoch: {} | Training Loss: {:.6f} | V
```

In [ ]:
```
plot_losses(train_losses_dense, valid_losses_dense, 'Training and Valid
```

In [ ]:
```
plot_aucs(valid_aucs_dense, 'Validation AUC over Epochs')
```

In [ ]:
```
print("Best learning rate according to hyperparameter search on Classic
```

In [ ]:
```
plot_losses(best_result['train_losses'], best_result['valid_losses'], '
```

In [ ]:
```
plot_losses(train_losses_dense, valid_losses_dense, 'Training and Valid
```

In [ ]:
```
plot_aucs(best_result['valid_aucs'], 'Validation AUC over Epochs - Clas
```

In [ ]:
```
plot_aucs(valid_aucs_dense, 'Validation AUC over Epochs - Dense Net')
```

In [ ]:
```
clear_memory()
```

In [ ]:
```
# turn of gradients
model_dense.eval()
preds = []

# iterate all test images
for batch_i, (data, target) in enumerate(test_loader):
    data, target = data.to(device), target.to(device)
    output = model_dense(data)

    pr = output.detach().cpu().numpy()
    for i in pr:
        preds.append(i)

# add predicted labels to submission file
df_sample_sub['label'] = preds
```

In [ ]:

```
# convert probabilities to float
for i in range(len(df_sample_sub)):
    df_sample_sub.label[i] = np.float(df_sample_sub.label[i])
```

In [ ]:

```
# create submission file
df_sample_sub.to_csv('submission.csv', index=False)
```