NAME: KINNERA HARSHITHA

RESEARCH ON IMAGE CLASSIFICATION MODEL BASED ON DEEP CONVOLUTION NEURAL NETWORK

1 | Introduction¶

1.1 | Why do we need this study?

- Microscopic evaluation of histopathalogic stained tissue & its subsequent digitalisation is now a more feasible due to the advances in slide scanning technology, as well a reduction in digital storage cost in recent years
- There are certain advantages that come with such digitalised pathology; including remote diagnosis, instant archival access & simplified procedure of consultations with expert pathologists
- Digitalised Analysis based on Deep Learning has shown potential benefits as a potential diagnosis tool & strategy
- Gulshan et al and Esteva et al demonstrated the **potential of deep learning for diabetic retinopathy screening** and **skin lesion classification**, respectively
- An essential task performed by pathologist; accurate breast cancer staging

- Assessment of the extent of cancer spread by histopathological analysis of sentinel axillary lymph nodes (SLNs) is an essential part of breast cancer staging process

1.2 | Problem Statement

- The sensitivity of SLN assessment by pathologists, however, is not optimal. A retrospective study showed that pathology review by experts changed the nodal status in 24% of patients.
- SLN assessment is **tedious** and **time-consuming**. It has been shown that deep learning algorithms could identify metastases in SLN slides with 100% sensitivity, whereas 40% of the slides without metastases could be identified as such
- This could result in a **significant reduction in the workload** of pathologists

1.3 | Study Aim

The aim of this study was to investigate the potential of using **PyTorch** Deep Learning module:

- For the **detection of metastases** in SLN slides and compare them with the predefined pathologist diagnosis.

linkcode

1.4 | Playground Prediction Competition

OVERVIEW

In this competition, you must create an algorithm to identify metastatic cancer in small image patches taken from larger digital pathology scans. The data for this competition is a slightly modified version of the PatchCamelyon (PCam) benchmark dataset (the original PCam dataset contains duplicate images due to its probabilistic sampling, however, the version presented on Kaggle does not contain duplicates).

PCam is highly interesting for both its size, simplicity to get started on, and approachability. In the authors' words:

[PCam] packs the clinically-relevant task of metastasis detection into a straight-forward binary image classification task, akin to CIFAR-10 and MNIST. Models can easily be trained on a single GPU in a couple hours, and achieve competitive scores in the Camelyon16 tasks of tumor detection and whole-slide image diagnosis. Furthermore, the balance between task-difficulty and tractability makes it a prime suspect for fundamental machine learning research on topics as active learning, model uncertainty, and explainability.

2. | Target feature class balance

Definitely not as one sides as was expected:

- The dataset favours **non- malignant**, normal cases (13k)
- compared to **non-malignant** cases (8.9k)

2.1 | Dataset preview

Let's also visualise the dataset images:

- **non-malignant** cases (0) (outligned with green colour)
- **malignant** cases (1) (outligned with red colour)
- We can note that its quite a **challenge to distinguish** whether an image should be classified as **malignant** or **non-malignant** simply from an inspection
- An **expert evaluation** is quite beneficial:
    - However it is likely a very **time consuming procedure** as indicated in the introduction

| Data Preparation¶

3.1 | Custom dataset class

- Let's create a custom Dataset class by subclassing the Pytorch Dataset class:
    - We need just two essential fuctions __len__ & __getitem__ in our custom class
- To speed up the training process, were using only 4000 samples for the entire dataset

4 | Splitting the Dataset¶

4.1 | random_split

- Among the training set, we need to evaluate the model on validation datasets to track the model's performance during training.
- Let's use 20% of img_dataset for **validation** & use the rest as the **training** set, so we have a 80/20 split

5 | Image Augmentation Definitions IMAGE AUGMENTATIONS

- Among with pretrained models, image **transformation** and **image augmentation** are generally considered to be an essential parts of constructing deep learning models.
- Using image transformations, we can expand our dataset or resize and normalise it to achieve better model performance.
- Typical transformations include **horizontal**,**vertical flipping**, **rotation**, **resizing**.
- We can use various image transformations for our binary classification model without making label changes; we can flip/rotate a **malignant** image but it will remain the same, **malignant**.
- We can use the torchvision module to perform image transformations during the training process.

TRAINING DATA AUGMENTATIONS

- transforms.RandomHorizontalFlip(p=0.5): Flips the image horizontally with the probability of 0.5
- transforms.RandomVerticalFlip(p=0.5) : Flips the image vertically "
- transforms.RandomRotation(45) : Rotates the images in the range of (-45,45) degrees.
- transforms.RandomResizedCrop(96,scale=(0.8,1.0),ratio=(1 .0,1.0)) : Randomly square crops the image in the range of [72,96], followed by a resize to 96x96, which is the original pixel size of our image data.
- transforms.ToTensor() : Converts to Tensor & Normalises as shown above already.

6| Optimiser Definition¶

- Training the network involves passing data through the network:
  - Using the **loss function** to **determine the difference between the prediction & true value**
  - Which is then followed by using of that info to **update the weights** of the network

- In an attempt to **make the loss function return as small of a loss as possible, performing updates on the neural network**, an **optimiser** is used

- The torch.optim contains implementations of common optimisers
- The **optimiser** will **hold the current state and will update the parameters based on the computed gradients</mark>**
- For binary classification taskss, **SGD**, **Adam** Optimisers are commonly used, let's use the latter here.

7 | Inference

- Once we have trained our model using train_val, we can begin to utilise it to **make some predictions**
- We have a whole dataset of **unlabelled image** data in folder test
- The unique ids of each image in the dataset are located in file sample_submission.csv • Like for thr training dataset, well create a data loader, using only tensor transformation
- As we have no label data, we need a slightly modified data class CODE:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
from plotly.subplots import make_subplots
import plotly.graph_objs as go
import copy
import os
import torch
from PIL import Image
from PIL import Image, ImageDraw
from torch.utils.data import Dataset
import torchvision.transforms as transforms
from torch.utils.data import random_split
from torch.optim.lr_scheduler import ReduceLROnPlateau
import torch.nn as nn
from torchvision import utils
%matplotlib inline
```

```
# library which allows us to view model summary like keras/tf
!pip install torchsummary
```

```
Collecting torchsummary
  Downloading torchsummary-1.5.1-py3-none-any.whl (2.8 kB)
Installing collected packages: torchsummary
Successfully installed torchsummary-1.5.1
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting be
haviour with the system package manager. It is recommended to use a virtual environment ins
tead: https://pip.pypa.io/warnings/venv
```

```
from IPython.core.display import display, HTML, Javascript

color_map = ['#FFFFFF','#FF5733']

prompt = color_map[-1]
main_color = color_map[0]
strong_main_color = color_map[1]
custom_colors = [strong_main_color, main_color]

css_file = '''
div #notebook {
background-color: white;
line-height: 20px;
}

#notebook-container {
%s
margin-top: 2em;
padding-top: 2em;
border-top: 4px solid %s;
```

```
-webkit-box-shadow: 0px 0px 8px 2px rgba(224, 212, 226, 0.5);
    box-shadow: 0px 0px 8px 2px rgba(224, 212, 226, 0.5);
}

div .input {
margin-bottom: 1em;
}

.rendered_html h1, .rendered_html h2, .rendered_html h3, .rendered_html h4, .rendered_html h5,
.rendered_html h6 {
color: %s;
font-weight: 600;
}

div.input_area {
border: none;
    background-color: %s;
    border-top: 2px solid %s;
}
```

```
.edit_mode div.cell.selected {
border-color: %s;

}
'''

def to_rgb(h):
    return tuple(int(h[i:i+2], 16) for i in [0, 2, 4])

main_color_rgba = 'rgba(%s, %s, %s, 0.1)' % (to_rgb(main_color[1:]))
open('notebook.css', 'w').write(css_file % ('width: 95%;', main_color, main_color, main_color_
rgba,
                                            main_color,  main_color, prompt, main_color, main_
color,
                                            main_color, main_color))

def nb():
    return HTML("<style>" + open("notebook.css", "r").read() + "</style>")
nb()
```

In [4]:
```
labels_df = pd.read_csv('/kaggle/input/histopathologic-cancer-detection/train_labels.csv')
print(labels_df.head().to_markdown())
```

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

In [5]:
```
os.listdir('/kaggle/input/histopathologic-cancer-detection/')
```

Out[5]:
```
['sample_submission.csv', 'train_labels.csv', 'test', 'train']
```

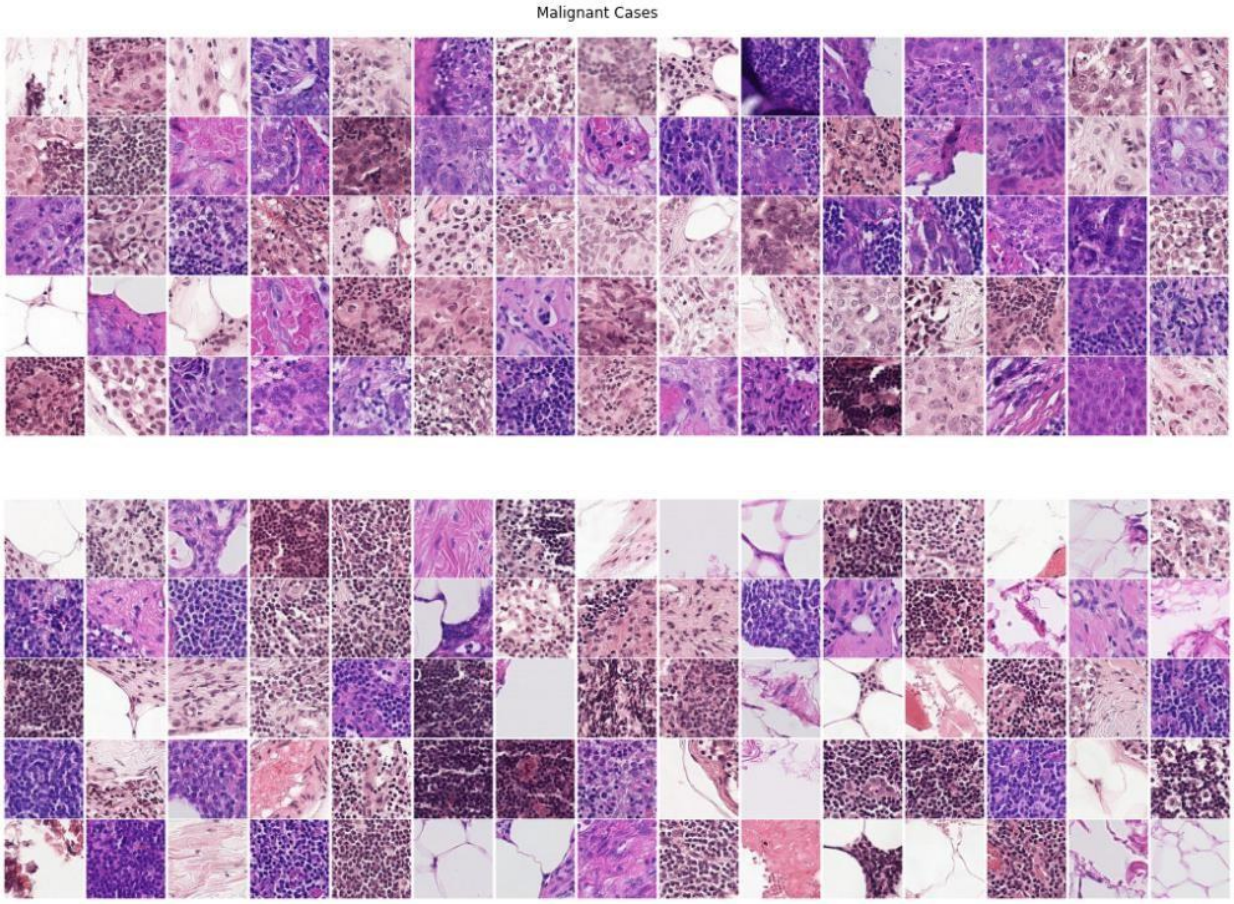
```
imgpath ="/kaggle/input/histopathologic-cancer-detection/train/"
his folder
malignant = labels_df.loc[labels_df['label']==1]['id'].values
es
normal = labels_df.loc[labels_df['label']==0]['id'].values
ses

print('normal ids')
print(normal[0:3],'\n')

print('malignant ids')
print(malignant[0:3])
```

```
normal ids
['f38a6374c348f90b587e046aac6079959adf3835'
 '755db6279dae599ebb4d39a9123cce439965282d'
 'bc3f0c64fb968ff4a8bd33af6971ecae77c75e08']
```

```
plot_fig(malignant,'Malignant Cases')
```



```
torch.manual_seed(0) # fix random seed

class pytorch_data(Dataset):

    def __init__(self,data_dir,transform,data_type="train"):

        # Get Image File Names
        cdm_data=os.path.join(data_dir,data_type)  # directory of files

        file_names = os.listdir(cdm_data) # get list of images in that directory
        idx_choose = np.random.choice(np.arange(len(file_names)),
                                      4000,
                                      replace=False).tolist()
        file_names_sample = [file_names[x] for x in idx_choose]
        self.full_filenames = [os.path.join(cdm_data, f) for f in file_names_sample]   # get t
he full path to images

        # Get Labels
        labels_data=os.path.join(data_dir,"train_labels.csv")
        labels_df=pd.read_csv(labels_data)
```

```
        # Get Labels
        labels_data=os.path.join(data_dir,"train_labels.csv")
        labels_df=pd.read_csv(labels_data)
        labels_df.set_index("id", inplace=True) # set data frame index to id
        self.labels = [labels_df.loc[filename[:-4]].values[0] for filename in file_names_sampl
e]  # obtained labels from df
        self.transform = transform

    def __len__(self):
        return len(self.full_filenames) # size of dataset

    def __getitem__(self, idx):
        # open image, apply transforms and return with label
        image = Image.open(self.full_filenames[idx])  # Open Image with PIL
        image = self.transform(image) # Apply Specific Transformation to Image
        return image, self.labels[idx]
```

```
# define transformation that converts a PIL image into PyTorch tensors
import torchvision.transforms as transforms
data_transformer = transforms.Compose([transforms.ToTensor(),
                                       transforms.Resize((46,46))])
```

```
# Define an object of the custom dataset for the train folder.
data_dir = '/kaggle/input/histopathologic-cancer-detection/'
img_dataset = pytorch_data(data_dir, data_transformer, "train") # Histopathalogic images
```

```
# load an example tensor
img,label=img_dataset[10]
print(img.shape,torch.min(img),torch.max(img))
```

```
torch.Size([3, 46, 46]) tensor(0.0415) tensor(0.9976)
```

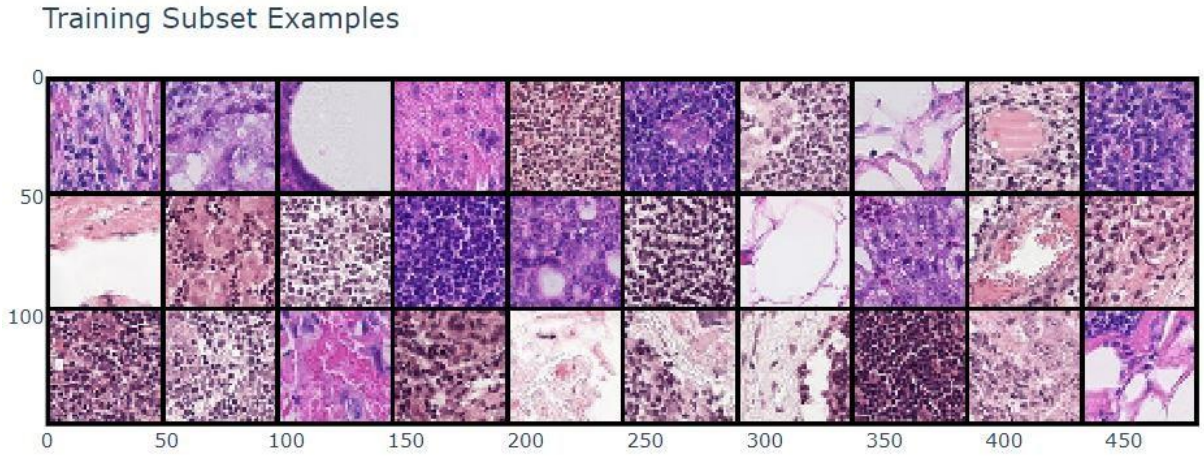
```
# Create grid of sample images
grid_size=30
rnd_inds=np.random.randint(0,len(train_ts),grid_size)
print("image indices:",rnd_inds)

x_grid_train=[train_ts[i][0] for i in rnd_inds]
y_grid_train=[train_ts[i][1] for i in rnd_inds]

x_grid_train=utils.make_grid(x_grid_train, nrow=10, padding=2)
print(x_grid_train.shape)

plot_img(x_grid_train,y_grid_train,'Training Subset Examples')
```

```
image indices: [1196 1346  789 2050 1296 3083 1363 2177 2034 1890 1662 1012 1575 2698
 2878  843 2173 3160  324 2697 1324 2198   59  125 2011  169 1131 2425
  650 2384]
```



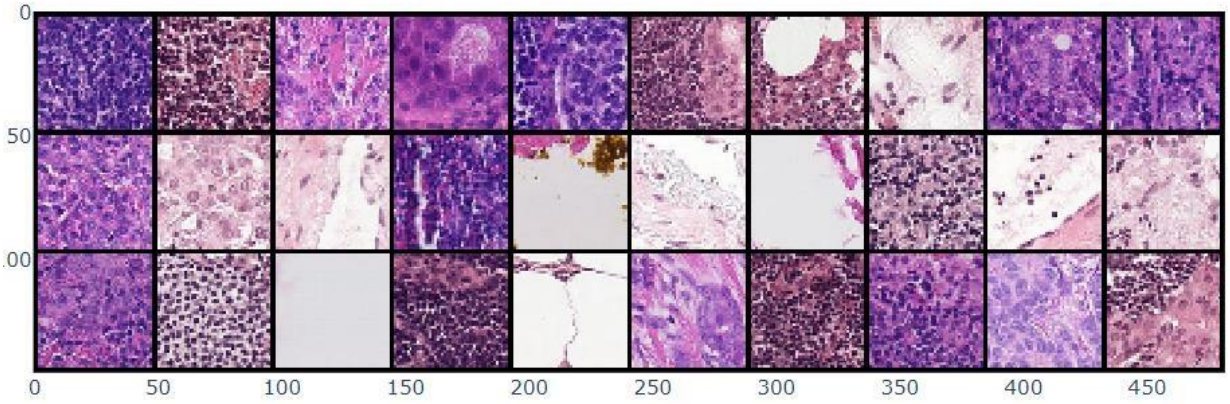

```
grid_size=30
rnd_inds=np.random.randint(0,len(val_ts),grid_size)
print("image indices:",rnd_inds)
x_grid_val=[val_ts[i][0] for i in range(grid_size)]
y_grid_val=[val_ts[i][1] for i in range(grid_size)]

x_grid_val=utils.make_grid(x_grid_val, nrow=10, padding=2)
print(x_grid_val.shape)

plot_img(x_grid_val,y_grid_val,'Validation Dataset Preview')
```

```
image indices: [217 320 458 665 237 260 702 519 234 277 136 123   3 412 170  61 786 786
 711 451 551 727 334 154 473 543 318  91 666 120]
torch.Size([3, 146, 482])
```

Validation Dataset Preview




```
# Define the following transformations for the training dataset
tr_transf = transforms.Compose([
#     transforms.Resize((40,40)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomVerticalFlip(p=0.5),
    transforms.RandomRotation(45),
#     transforms.RandomResizedCrop(50,scale=(0.8,1.0),ratio=(1.0,1.0)),
    transforms.ToTensor()])
```

```
# For the validation dataset, we don't need any augmentation; simply convert images into tensors
val_transf = transforms.Compose([
    transforms.ToTensor()])

# After defining the transformations, overwrite the transform functions of train_ts, val_ts
train_ts.transform=tr_transf
val_ts.transform=val_transf
```

```
# Neural Network Predefined Parameters
params_model={
        "shape_in": (3,46,46),
        "initial_filters": 8,
        "num_fc1": 100,
        "dropout_rate": 0.25,
        "num_classes": 2}

# Create instantiation of Network class
cnn_model = Network(params_model)

# define computation hardware approach (GPU/CPU)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = cnn_model.to(device)
```

```
from torchsummary import summary
summary(cnn_model, input_size=(3, 46, 46),device=device.type)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 8, 44, 44]             224
            Conv2d-2           [-1, 16, 20, 20]           1,168
            Conv2d-3            [-1, 32, 8, 8]            4,640
            Conv2d-4            [-1, 64, 2, 2]           18,496
            Linear-5                 [-1, 100]            6,500
            Linear-6                   [-1, 2]              202
================================================================
Total params: 31,230
Trainable params: 31,230
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.02
Forward/backward pass size (MB): 0.19
```

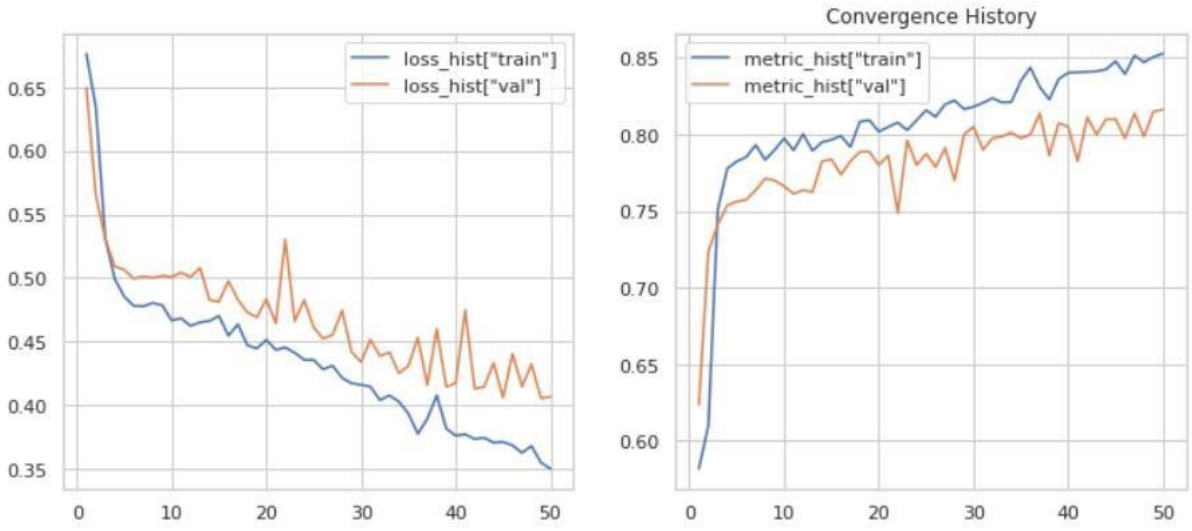
```
params_train={
 "train": train_dl,"val": val_dl,
 "epochs": 50,
 "optimiser": optim.Adam(cnn_model.parameters(),
                         lr=3e-4),
 "lr_change": ReduceLROnPlateau(opt,
                                mode='min',
                                factor=0.5,
                                patience=20,
                                verbose=0),
 "f_loss": nn.NLLLoss(reduction="sum"),
 "weight_path": "weights.pt",
 "check": False,
}
```

```
params_train={
 "train": train_dl,"val": val_dl,
 "epochs": 50,
 "optimiser": optim.Adam(cnn_model.parameters(),lr=3e-4),
 "lr_change": ReduceLROnPlateau(opt,
                                mode='min',
                                factor=0.5,
                                patience=20,
                                verbose=0),
 "f_loss": nn.NLLLoss(reduction="sum"),
 "weight_path": "weights.pt",
}

''' Actual Train / Evaluation of CNN Model '''
# train and validate the model

cnn_model,loss_hist,metric_hist=train_val(cnn_model,params_train)
```

```
Text(0.5, 1.0, 'Convergence History')
```



```
# class predictions 0,1
y_test_pred=np.argmax(y_test_out,axis=1)
print(y_test_pred.shape)
print(y_test_pred[0:5])
```

```
(57458,)
[0 0 0 1 0]
```

```
# probabilities of predicted selection
# return F.log_softmax(x, dim=1) ie.
preds = np.exp(y_test_out[:, 1])
print(preds.shape)
print(preds[0:5])
```

```
(57458,)
[0.13662416 0.04919121 0.19058716 0.64544386 0.2606593 ]
```