Capstone Project Report

Executive Summary

**I want to apply and test the ability of state-of-art techniques connecting text with image, zero-shot classification models, to predict or annotate unlabeled medical images, MedMNIST dataset which is a benchmark dataset used to test new algorithms or new methods. The general ImageNet zero-shot models can annotate the high-level labels, like chest or breast, with a high accuracy on this medical benchmark dataset. However, it performs poorly on granular-level labels like malignant, normal, benign and so on. We then train some traditional CNN models on one of dataset by using labels annotated by zero-shot model and fine-tune it to have the best accuracy on its test dataset.**

Introduction/background

**Because there are millions of unlabeled medical images generated in hospitals waiting for annotations before they can be downstream analyzed, but manually labeling them is too time consuming. Also, sometimes, the labels or classes that we want to predict are not in the all pretrained models or some of them are not. At this time, we come across zero-shot, one-shot or few-shot transfer learning.**

Data

**MedMNIST v2: A large-scale lightweight benchmark for 2D and 3D biomedical image classification. This benchmark dataset was collected and processed by 8 researchers from six institutions including universities and hospitals. They have published a paper about this dataset:** [**https://arxiv.org/abs/2110.14795**](https://arxiv.org/abs/2110.14795) **on the arxiv and they have a GitHub repository containing the guides and instruction of how to use this dataset by installing the package they wrote for this dataset, called medmnist. You can download each 2D and 3D respectively by clicking this website:** [**https://zenodo.org/record/5208230**](https://zenodo.org/record/5208230) **and also download all dataset automatically by setting download parameter in the package equal to True. The whole dataset is very easy to access. The researchers now are still actively supporting and developing this dataset and its GitHub Repo. Although dataset is not proposed for clinical use, it has many advantages of research purposes, such as diverse, standardize, lightweight, educational. The diverse dataset provides users with different organ images, different tasks, like binary/multi-class, regression problem.**

Methodology

**My goal is to automatically annotate the millions of unlabeled medical images without too much human involvement or manually labelling. I will use zero-shot clip model and convolutional neural network together to tackle this problem. The whole process involves two procedures. I first split data which we want to annotate into two parts. The first part of data will be annotated by zero-shot clip model choosing the highest probability label among all. Then I will use convolutional neural network to train the first part of data with their predictions from zero-shot clip model as their CNN ground truth labels. Eventually, I will evaluate this trained CNN model on the second part of the data with their truly ground labels. Once I have this best tuned CNN model, I am able to leverage this CNN to annotate new unlabeled medical images. In this way, I can show that using two method together can achieve a higher accuracy on annotation than only using zero-shot clip model.**

Zero-shot clip model

**I will first introduce the state-of-the-art model which connects text and images, clip model. The clip model efficiently learns visual concepts from natural language supervision. Clip model can be applied to any visual classification task by simply giving the visual labels to be recognized. The clip model is trained on a very wide range of natural language supervisions and images on the internet. The general clip model performs better than ImageNet Resnet101 on ImageNet, ImageNet v2, ImageNet Rendition, ObjectNet, ImageNet Sketch, ImageNet Adversarial.**

**Clip is the short name for Contrastive Language Image Pretraining. As the name suggests, Clip model is trained to pair the natural language concepts with the image concepts. Clip models want to realize the prediction and pair the image concepts with unseen labels. This is very impressive and inspirational words, because before this idea and clip model, in computer vision, people always train and predict labels on observed/seen labels. However, as demand increases, there are many unseen/unobserved categories needed to be recognized in images.**

**The overall architecture of clip models is below:**

**Diagram

Description automatically generated**

**The model contains three phases. The first phase is to train two encoders, the first is text encoder which extracts and learns the text representations in T1, T2, …, TN; the second is image encoder that learns visual representations in I1, I2, …, IN. Then the dot product or other similarity score between paired text representations and visual representations is maximized while that of non-paired text representation and visual representations is minimized. After this optimization, captions that warp the unseen labels are fed into text encoder to produce each text representation for each class, and the unrecognized image is fed into image encoder to produce its visual representation. Finally, computing cosine or other similarity score between this visual representation and each class text representation gives the highest score, and its corresponding class will be chosen as the prediction label for the new image. The above is overall process for training and prediction.**

**The trained clip model perform well on some common dataset, like 90.1% accuracy on FOOD101 data, 90.2% accuracy on SUN397, 89.0% accuracy on YOUTUBE-BB.**

**There are some advantages of this kind of zero-shot model, the first is CLIP is highly efficient.**

**Chart, line chart

Description automatically generated**

**The second one is CLIP is flexible and general, because the CLIP is trained on a wide range of visual and text dataset.**

Chart, line chart

Description automatically generated

Convolutional Neural Network

**The next architecture I apply is the convolutional neural network. This is very famous artificial neural network in image analysis like image classification, image segmentation, natural language processing. Recent year, there are too many research and applications in CNN because it is very powerful and useful in image analysis, including video analysis, signal analysis.**

**As the below figure suggests, the input digit image 2 is the input layer: (28\*28\*1). In this CNN architecture, it contains two convolutional layers, two max-pooling layers, two fully connected layers. The first convolutional layer has n1 filter/kernels, each of which is 5 by 5. Each filter is sliding over the width and height of the input layer and compute the dot product between the selected input layer entries and this filter map, once this filter slide over all possible width and height, it will produce a 2-dimensional activation map. If the first convolutional has n1 filters, then before the next layer, there are n1 activation maps. Each filter map will learn the different visual pattern of the input layer, such as edges of the orientation, the colors, and so on. In the below example, the input layer only has one channel (1-dimensional depth), however, in reality, most images have three channels (3-dimensional depth), in this case, the filter map should also be 3-dimensional depth to make each activation map only 2-dimensional space. In other words, the depth of filter map is always equal to the depth of the input layer. The most important property of CNN is the local connectivity and shared weights, because the image dimension is huge, making the fully connectivity impractical. The same filter will slide over all width and height of the input layer so weights from this filter map will be used to compute dot product with receptive field of the input layer for many times. The fully connectivity and shared weights can significantly reduce the computational cost and learn the local and different visual patterns. The filter map size usually is small, 3 by 3 and 5 by 5 are the most common one. The next most common layer is the Max-Pooling, here is 2 by 2 max-pooling map with 2 strides. Max-pooling layers can dramatically reduce the spatial size of the input layer so that the number of parameters is reduced as well to make computations efficient. The bonus advantage of it is to avoid the overfitting because overfitting often happens in the deep neural network due to too many parameters to fit the noise. By using 2 by 2 max-pooling layer with 2 strides, the 24 by 24 activation map is reduced to 12 by 12. At the end of this architecture, there are two fully connected layers companied with a non-linear ReLU activation layer. There is also dropout layer in the last one. The dropout layer is to mask out some neurons in this layer by a probability like 0.5 to overcome the common issue, overfitting in deep neural network. The final layer contains 10 neurons, each of which presents a score/probability for each class.**

Diagram

Description automatically generated

**The below picture is the typical local connectivity, shared weights in one convolutional layer. The input has three channels and one padding, there are two filters, each of which is 3 by 3 by 3, so the output volume has two activation maps.**

Diagram

Description automatically generated

**In practice, people don’t need to design a CNN architecture from scratch, they first should look for the current state-of-the-art pretrained models, like AlexNet, VGGNet, ResNet, and use these pretrained models to fine-tune on their own dataset. The best CNN model on ImageNet usually works better than the people’s own customized model.**

My workflow of combinational two above models

**Our problem is that we want to automatically annotate the medical images without manually labelling. So, I will apply one of best zero-shot clip models to predict the unseen image labels. However, these prediction labels only have 80% accuracy. Thus, the second stage is to train a custom CNN model to improve this prediction accuracy to 95% accuracy.**

**There are currently 8 pretrained zero-shot models:**

import clip

clip.available\_models()

['RN50',

'RN101',

'RN50x4',

'RN50x16',

'RN50x64',

'ViT-B/32',

'ViT-B/16',

'ViT-L/14']

**After some experiments and comparisons, the Vision Transformers-L/14 (ViT-L/14) performed best among others. ViT-L/14 was chosen to be used. It was pretrained on ImageNet containing around 14 million images with 21,843 classes, then fine-tuned on another ImageNet at resolution 224 by 224. Next, we can load this huge pretrained vision transformer model to have a look at its overall sketch.**

model, preprocess = clip.load("ViT-L/14")

model.cuda().eval()

input\_resolution = model.visual.input\_resolution

context\_length = model.context\_length

vocab\_size = model.vocab\_size

print("Model parameters:", f"{np.sum([int(np.prod(p.shape)) for p in model.parameters()]):,}")

print("Input resolution:", input\_resolution)

print("Context length:", context\_length)

print("Vocab size:", vocab\_size)

Model parameters: 427,616,513

Input resolution: 224

Context length: 77

Vocab size: 49408

**From the above code and results, we can observe that there are 427 million model parameters, the input resolution is 224 consistent with the pretrain ImageNet resolutions, the context length 77 refers to the maximum text size fed into the model. Training over 427 million model parameters requires a lot of computational resources and time, luckily, we don’t need to train this kind of huge model, all we need is just apply this model on medical images in the following.**

**I will use two benchmark datasets as illustration to my method. The first one is Chest MNIST which contains 78,468 training samples, 11,219 validation samples, 22,433 test samples. The second one is Breast MNIST which contains 546 training samples, 78 validation samples, 156 test samples. I chose all 546 breast training samples and randomly chose 546 chest training samples to form a mixture of two benchmark datasets. The following picture is 9 random images with its true labels selected from this merged dataset.**

half\_sample\_size = 546

random\_chest\_inx = np.random.choice(chest['train\_images'].shape[0], size=half\_sample\_size, replace=False)

random\_breast\_inx = np.random.choice(breast['train\_images'].shape[0], size=half\_sample\_size, replace=False)

data = np.concatenate([breast['train\_images'][random\_breast\_inx], chest['train\_images'][random\_chest\_inx]], axis=0)

true\_labels = [0]\*half\_sample\_size + [1]\*half\_sample\_size

data.shape

(1092, 28, 28)

**Graphical user interface, application

Description automatically generated**

**Next, I will create two captions encompassing these two labels, chest and breast. “This is a photo of chest, and it’s a type of organ” and “This is a photo of breast, and it’s a type of organ.” For CLIP model, it’s typically to write a short caption or sentence capturing the labels then feed these captions into text encoder model to produce normalized text features like below code:**

uni\_labels = ['breast', 'chest']

text\_descriptions = [f"This is a photo of {label}, and it's a type of organ." for label in uni\_labels]

text\_tokens = clip.tokenize(text\_descriptions).cuda()

with torch.no\_grad():

text\_features = model.encode\_text(text\_tokens).float()

text\_features /= text\_features.norm(dim=-1, keepdim=True)

**Then, I will convert all NumPy-array images to tensor format so that PyTorch-based framework CLIP vision transformer model can process as well as moving all tensor to GPU cuda memory to speed up computation efficiency. After these data preprocessing, using image encoder to creates all image features for all images. Lastly, I will use the cosine similarity to compute the similarity between all previous text features and their associated image features to find the max probability label score, then assign that label as each prediction. The following code illustrate all these simple processes:**

SS = data.shape[0]

original\_images = [Image.fromarray(data[j]) for j in range(SS)]

images = [preprocess(j) for j in original\_images]

image\_input = torch.tensor(np.stack(images)).cuda()

with torch.no\_grad():

image\_features = model.encode\_image(image\_input).float()

text\_probs = (100.0 \* image\_features @ text\_features.T).softmax(dim=-1)

top\_probs, top\_labels = text\_probs.cpu().topk(1, dim=-1)

**After I get the prediction labels, I am able to compute the prediction accuracy by two lines of codes as following:**

pred\_labels = np.squeeze(top\_labels).numpy()

(true\_labels == pred\_labels).sum() / len(true\_labels)

0.804945054945055

**Eventually, by using this vision transformer CLIP model, I got around 80% accuracy on this 1092 training samples in binary classification.**

**Once I have these prediction labels, I will train a convolutional neural network on these prediction labels to improve the accuracy. The magic and powerful thing is that this trained CNN model from scratch can correct some mislabeling in the CLIP model predictions and then improve the accuracy a lot.**

**First, I must design this convolutional neural network. Since the breast and chest medical images are relatively small 28 by 28 and there are only two classes waiting for being recognized, the CNN model doesn’t have to be deep, namely a shallow CNN model is capable of classifying binary cases. So, I designed two convolutional layers followed by two the most common max pooling layers. The ReLU non-linear activation functions will be applied after each convolutional layer. After these, a flatten layer from images will go through three fully connected linear layers companied by three times dropout operations to overcome overfitting issues. Finally, the two-neuron output layer will be left to produce the probability score. This designing architecture is shallow and simple but performs very well. We don’t need huge pretrained CNN model like AlexNet, VggNet to achieve the same effect. The following picture displays the details of my customized Convolutional Neural Network.**

Chart, line chart

Description automatically generated

**The following code snippet is the PyTorch-based convolutional neural network I used to train which is the same as described above.**

class bmodel(nn.Module):

def \_\_init\_\_(self):

super(bmodel, self).\_\_init\_\_()

self.conv1 = nn.Conv2d(1, 10, 3)

self.pool = nn.MaxPool2d(kernel\_size=2, stride=2)

self.conv2 = nn.Conv2d(10, 20, 4)

self.fc1 = nn.Linear(20 \* 5 \* 5, 250)

self.fc2 = nn.Linear(250, 50)

self.fc3 = nn.Linear(50, 2)

self.dropout = nn.Dropout(0.5)

def forward(self, x):

x = self.pool(F.relu(self.conv1(x)))

x = self.pool(F.relu(self.conv2(x)))

x = torch.flatten(x, 1)

x = self.dropout(x)

x = F.relu(self.fc1(x))

x = self.dropout(x)

x = F.relu(self.fc2(x))

x = self.dropout(x)

x = self.fc3(x)

return x

**Usually, we will move all our data and model to the GPU cuda memory to speed up the computations and make use of our available resources. Cuda is Compute Unified Device Architecture, which can achieve parallel computing. It will improve your learning speed in your parameter update by using GPU rather than CPU.**

# Check whether there is a gpu for cuda

train\_on\_gpu = cuda.is\_available()

print(f'Train on gpu: {train\_on\_gpu}')

# Number of gpus

if train\_on\_gpu:

gpu\_count = cuda.device\_count()

print(f'{gpu\_count} gpus detected.')

if gpu\_count > 1:

multi\_gpu = True

else:

multi\_gpu = False

else:

multi\_gpu = False

print(train\_on\_gpu,multi\_gpu)

if train\_on\_gpu:

model = model.to('cuda')

Train on gpu: True

1 gpus detected.

True False

**After reading in the previous merged dataset with 546 chest images, 546 breast images and their zero-shot CLIP predictions, I am going to randomly split this merged dataset into 80% training and 20% validation set in the following code chunk:**

train\_ind = np.random.choice(all\_ex\_data.shape[0], int(all\_ex\_data.shape[0] \* 0.8), replace=False)

valid\_ind = np.setdiff1d(range(all\_ex\_data.shape[0]), train\_ind)

train\_X = all\_ex\_data[train\_ind]

train\_Y = all\_labels[train\_ind]

validation\_X = all\_ex\_data[valid\_ind]

validation\_Y = all\_labels[valid\_ind]

Train data shape: (873, 1, 28, 28)

Train labels shape: (873,)

Validation data shape: (219, 1, 28, 28)

Validation labels shape: (219,)

**Next, I will transfer the images from NumPy to Tensor and place the images in the data loader to shuffle data and extract the batch size of image to update the gradients. The following code chunk illustrates this:**

batch\_size = 4

data = {

'train':

TensorDataset(torch.from\_numpy(train\_X), torch.from\_numpy(train\_Y).float()),

'valid':

TensorDataset(torch.from\_numpy(validation\_X), torch.from\_numpy(validation\_Y).float())

}

dataloaders = {

'train': DataLoader(data['train'], batch\_size=batch\_size, shuffle=True,num\_workers=10),

'valid': DataLoader(data['valid'], batch\_size=batch\_size, shuffle=True,num\_workers=10)

}

# Iterate through the dataloader once

trainiter = iter(dataloaders['train'])

validationiter = iter(dataloaders['valid'])

**As we know, the most important hyperparameter in the deep learning is the gradient update method, learning rate, and the type of loss function. In my case, because it’s binary classification problem, the most suitable loss function is cross entropy function. I choose the general conventional gradient update method, Stochastic Gradient Descent and set the learning rate 0.001, momentum equal to 0.9 like the following:**

criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

**Once I set up all these environments and image processing, I start training the convolutional neural network. In the training function, I use the validation data loader to for maximum number of epochs with no improvement in validation loss for early stopping. I set this argument to 10 and maximum number of training epochs to 20. Then the following code specifies the training function setup:**

model, history = train(model,

criterion, optimizer,

dataloaders['train'], dataloaders['valid'],

save\_file\_name=save\_file\_name, max\_epochs\_stop= 10, n\_epochs=20,

print\_every=1)

**After this training process, I plot the following training and validation loss and accuracy curves against each epoch:**

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

**As we can observe from these two plots, training and validation loss continue decreasing from the first epoch until the last one on average, however the curves are oscillation a lot. The training accuracy and validation accuracy continue increasing after 5 epoch and then keep in the flat level, and the validation accuracy are greater than the training accuracy for most epoch except 12.5 and 17.5. In general, my two convolutional layer CNN model perform well on this task.**

**Next, I will evaluate how this combination of zero-shot CLIP model and CNN model improve the accuracy on the held-out test set.**

**First, let’s get the prediction on the validation test before computing generalized out-of-sample accuracy,**

dataiter = iter(validationiter)

# get some random training images

images, labels = dataiter.next()

# Get the prediction of images by using your model.

outputs = model(images.cuda().float())

\_, predicted = torch.max(outputs, 1)

# print images

imshow(torchvision.utils.make\_grid(images))

print('GroundTruth: ', ' '.join('%5s' % classes[labels[j].long()] for j in range(batch\_size)))

print('Prediction: ', ' '.join('%5s' % classes[predicted[j].long()] for j in range(batch\_size)))

A picture containing graphical user interface

Description automatically generated

**As we can observe from the above picture, the ground truth labels are the zero-shot CLIP model predictions. Obviously, the third and fourth image should be chest, however, the zero-shot classifies them into breast. However, the powerful and magic thing is this CNN trained model can correct the mislabeling in the zero-shot predictions. As we see, the third and the fourth images are classified as chest correctly.**

**Finally, I will evaluate this CNN model’s performance on the held-out test set. I choose all 234 breast images and random 234 chest images from the larger pool. In total, 468 images are there.**

test\_channel = np.expand\_dims(test, axis=1)

test\_channel.shape

(468, 1, 28, 28)

**After creating the true labels for this merged test set, I will compare true labels against the CNN model prediction:**

outputs = model(torch.from\_numpy(test\_channel).cuda().float())

\_, predicted = torch.max(outputs, 1)

(np.array(true\_labels) == np.array(predicted.tolist())).sum() / len(true\_labels)

**0.9572649572649573**

**I get the 95.72% accuracy on 468 test set. This shows that my idea and new method works perfectly on this binary classification medical image problem. This CNN model can be used to automatically annotate the new generated chest and breast images without any human manually labelling.**

Results

Conclusion/Next Steps

**There are some pretrained medical image deep learning models, like in the monai, medical open network for artificial Integillence which provides domain-optimized foundational capabilities for developing healthcare imaging training workflows in native PyTorch language. In the future, I may plan to embed them into current zero-shot, N-shot transfer learning architecture then apply them into MedMNIST image dataset because current N-shot learning is trained on ImgaeNet which is far off medical fields.**

**Once the open AI releases their codes which are used to trained the their zero-shot classification model, I plan to use those codes to train on all medical dataset, like all MedMNIST benchmark dataset and fine-tune it. Then I can have a medical domain adaptation zero-shot models. Then I test and apply them to some other medical dataset to annotate them and evaluate the results. In this case, the results should be much better than the general ImageNet -based zero-shot’s performance.**