Scoliosis Subteam Midterm Presentation

Eric Chen, Nathan Zhong, Lucas Yim, Brian Zhang, Patrick Weng, Daniel Yu, Charlotte Hettrich, Ruchi Patel, Aditya Chandaliya, Yahya Hassan, Minseung Jung, Pujith Veeravelli, Pranav Malireddy

Overview

- Introduction
- Azure
- Semester Work
- Datasets
- EMADE
- Future Work

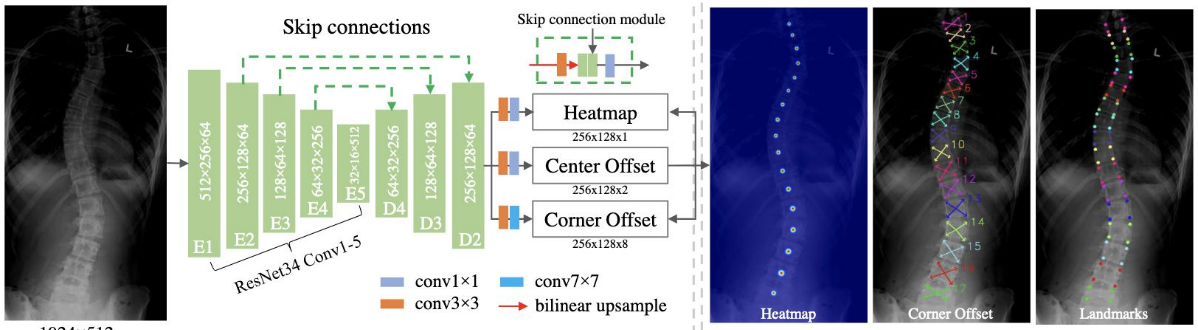
What is Scoliosis and Scoliosis Detection?

Introduction

- Adolescent Idiopathic Scoliosis (AIS) is a common form of scoliosis affecting children ages 10-18, which can lead to pain and further medical complications.
- Standard measure of Scoliosis is the cobb angle, which is the angle created by the most tilted pair of vertebrae.
    - An angle of at least 10 degrees is defined as scoliosis.
    - Angles between 10 and 20 degrees require close monitoring from a doctor
    - Angles between 20 and 40 degrees, can necessitate back braces.
    - Beyond 40 degrees, surgery is generally required.
    - Between professionals, recorded angles can vary from 2-10 degrees. This inconsistency is exacerbated by patient positioning.

Goal

- We want to determine the cobb angle of a set of x-ray spinal images quickly and accurately.
- Several vertebra detection and cobb angle detection challenges exist; we considered several challenge winners and research papers on challenge datasets.
- Using the Vertebra-Focused Landmark Detection for Scoliosis Assessment (VFL) paper
  - Proposed a method predicting cobb angles based on identifying centerpoints of vertebrae, then finding corresponding corner points, and finally calculating cobb angle.



(a) Vertebra-Focused Landmark Detection Network (b) Landmark Decoding

Loss Metrics/Evaluation Functions

- SMAPE (Symmetric Mean Squared Error)
    - The percentage error can be used to estimate the degrees by which the cobb angle is off
    - Scale invariant: Measures models that use different scales and units
    - Symmetric: Treats positive and negative values equally

$$\text{SMAPE} = \frac{100\%}{n} \sum_{t=1}^{n} \frac{|F_t - A_t|}{|A_t| + |F_t|}$$

- MSE (Mean Squared Error)
    - Measures the average of the squares of the errors to evaluate fitness of landmark detection
    - Emphasizes large errors by squaring them and less sensitive to small deviations between predicted and actual

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2.$$

Azure

Azure

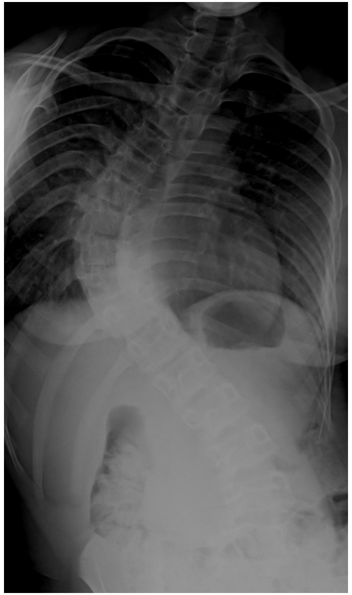
- Sponsored by Shriners
- Our subteam uses Microsoft Azure instead of PACE
- We use AzureSQL, Azure Compute, and Azure Data Storage

Azure SQL

- Emade typically runs on MariaDB/MySQL
- Since we switched branches from Cache-V2 to Athite3-nn-final, we needed to modify it to work with our existing AzureSQL database
- Added functionality to specify in template file what type of database

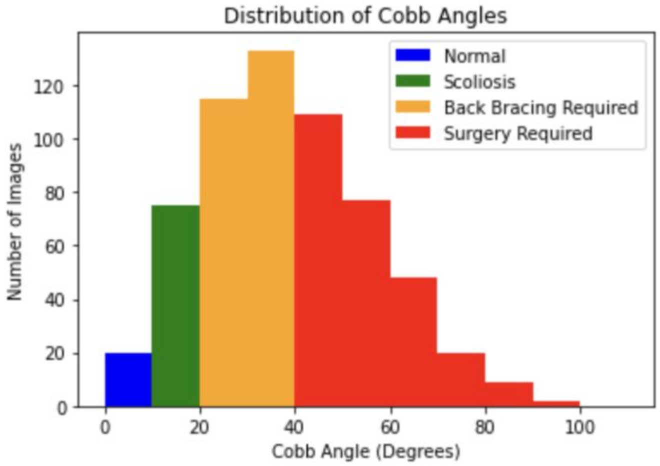
Datasets

Examples



Shriner’s Dataset

- Collection of x-rays provided by the Shriners Hospitals for Children. Images will be used for evaluation of individual algorithms produced.
- Images tend to be less uniform and have more variation than the AASCE dataset
    - Running the AASCE model on the Shriner’s data initially with no preprocessing resulted in very bad predictions
- We were able to replicate the results of the VFL model on the AASCE dataset and make predictions on Shriners images. Runs of the model on Azure yielded a SMAPE of 9.06 on the AASCE dataset and 16.5 on the Shriners image set.



Preprocessing - Cropping

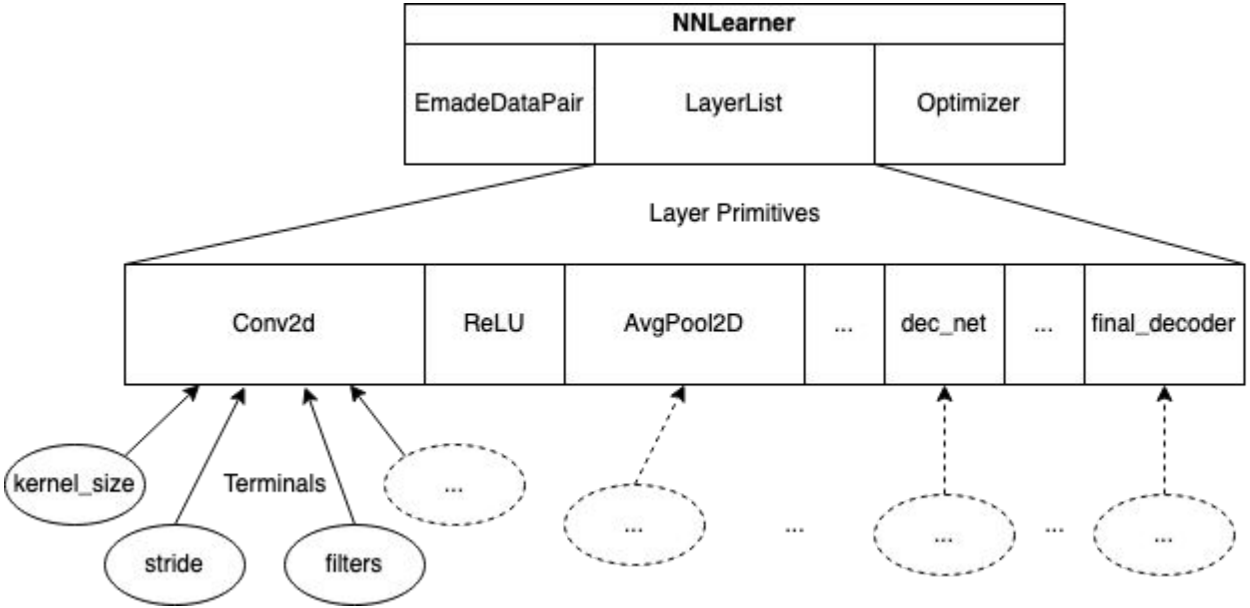
- Noticed that the SMAPE and MSE of the landmarks for the Shriner's cropped images performed no better than beforehand
    - This was odd since when visualizing the landmarks on the cropped Shriner's images it did seem to perform better
- Realized that after cropping, never adjusted the landmarks so the points were still formatted for the uncropped images
    - Updated landmarks and cropped images housed in
- SMAPE Pre-Cropping: 16.5 -> SMAPE Post-Cropping 11.72
- Issue that still needs to be handled are Shriner's xrays which are inverted in colors which the AASCE model in itself cannot predict well on

Preprocessing - NPZs

- Our previously packed AASCE and Shriners npz’s threw errors when inputted to resnet
  - Had to repack the images
- Changed truth data to landmarks instead of cobb angle
- Resampled the images and remapped truth data to consistent size since we ran into errors with ragged tensors

Semester Work

Individual Structure



ResNet

- Existing AASCE model architecture -> ResNet
- Generally consists of generic feed-forward layers with skip connections
- How can we come up with a better architecture? EMADE!
- Utilize components as primitives
    - Residual blocks
    - Pooling layers
    - Convolutional layers

NNLearner

- Last semester: Using EMADE to evolve on the pipeline of preprocessing primitives into a single model primitive
    - Flaw: Not much to evolve on and make new discoveries since it's just determining the best combination of some number of preprocessing with no evolution within the actual model
- This semester: Transitioned to using NNLearner in order to actually develop and evolve a model using EMADE
- Capable of building models out of ResBlocks and individual layers with NNLearner
    - Common format to add layers/parameters to layerlist in our EMADE in Azure
- Looking to get full EMADE run to develop EMADE individuals with the functioning NNLearner primitive

```
ARG0 = self.image_data
llist = nnm.InputLayer()
llist = nnm.ZeroPadding2DLayer(3, llist)
llist = nnm.OutputLayer(llist)
result = nnm.NNLearner(ARG0, llist,'adam')
```

```
layerlist:  ['input', ('zeropadding2d', (3, 3)), 'output']
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 500, 500, 1)]     0
_________________________________________________________________
zero_padding2d (ZeroPadding2 (None, 506, 506, 1)       0
_________________________________________________________________
```

```
pset.addPrimitive(nnm.NNLearner, [EmadeDataPairNN, nnm.LayerListFinal, nnm.Optimizer], EmadeDataPairNNF, name="NNLearner")
```

DecNet

- DecNet is organized into three combination modules which decode the ResNet block output.
    - self.up = nn.Sequential
        - nn.Conv2D(c_low, c_up, kernel_size=3, padding=1, stride=1))
        - nn.BatchNorm2D(c,up)
        - nn.ReLU(inplace=True)
    - self.cat_conv = nn.Sequential
        - nn.Conv2D(c_up*2, c_up, kernel_size=1, stride=1)
        - nn.BatchNorm2D(c_up)
        - nn.ReLU(inplace=True)
- DecNet.py uses the combinational module layers and returns a dictionary
- Most layers are common Keras layers already implemented in EMADE but wrote two custom layers
    - CustomImageResizing -> Similar to Keras Resizing Layer but takes in second image which provides dimensions
    - CustomDecNetLayer -> Takes in input and outputs a dictionary corresponding to heatmap, wh, and reg

Decoder

- Takes in the dictionary of decoder outputs (hm, wh, reg) and “decodes” them into landmark form
- Outputs 17 sets of 4 points which represent the corners of each of the 17 landmarks
- Successfully converted decoder into Keras with little to no error
  - Hardcoded certain values that would have been alterable in the command line when calling AASCE
  - These include confidence threshold, image size (Since we are standardizing in in the npz), and the number of vertebra
- Implemented in EMADE as a custom layer (That will not have trainable weights) which we can add to the layerlist as a part of the pipeline
  - Necessary since the npz is packaged to have landmarks as truth data, which can only be evaluated by having the decoder turn raw data into usable landmark data

PyTorch to Keras Conversions

- The VFL Model has 4 PyTorch layers: nn.Conv2d, nn.BatchNorm2d, nn.ReLU, and nn.MaxPool2d.
- Most of the parameters of the Keras equivalents are similar, with just a few differences:
  - Padding as a parameter is not as flexible in Keras, must use ZeroPadding2D() layer to have int/tuple padding.
  - Keras is by default channels last, while PyTorch is channels first. To maintain compatibility data_format = "channels-first" should be specified.
- We created a testing script to verify that comparisons match.
  - The script takes in a random numpy array of any dimension, and feeds it to a Keras and a PyTorch model.
  - Compares the output at the end, and checks to see that there is only small differences between them (~ 1e-5).
  - Note: must make sure both models are in the same "mode" .
    - PyTorch is by default training mode.
    - Keras is by default in eval mode.

```
# CONV2D Test
test_torch_keras_conversion(
    [nn.Conv2d(in_channels = 3, out_channels = 10, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias = False)],
    [keras.layers.ZeroPadding2D(padding=(3,3)), keras.layers.Conv2D(filters=10, kernel_size=(7, 7), strides=(2, 2), use_bias=False)],
    (1, 64, 64, 3)
)
```

PyTorch to Keras Conversions

PyTorch Model

```
import torch.nn as nn
from torchinfo import summary

class NeuralNet(nn.Module):
  def __init__(self):
    super(NeuralNet, self).__init__()
    self.hidden1 = nn.Conv2d(64, 64, kernel_size=(7, 7), stride=(2, 2), padding = (3, 3), bias = False)
    self.hidden2 = nn.BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    self.hidden3 = nn.ReLU(inplace=True)
    self.hidden4 = nn.MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)

model = NeuralNet()
summary(model)
```

Keras Model

```
from tensorflow import keras

height = 1
width = 1

# we do not know the input shape
model = keras.Sequential()
model.add(keras.layers.Conv2D(filters=64, kernel_size=(7, 7), strides=(2, 2), padding="same", use_bias=False,
model.add(keras.layers.BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05, center=True, scale=True))
model.add(keras.layers.Activation('relu'))
model.add(keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2), padding="same", data_format="channels_last")

print(model.summary())
```

PyTorch to Keras Conversions

- We wanted to build out the entire VFL model in Keras to compare loss and performance over epochs.
- The training script for VFL's PyTorch version depends on too many PyTorch dependencies.
    - We did not have time to create and test a Keras equivalent.
    - We plan to convert most of these dependencies into Keras–namely, the way data is loaded must be done with a Keras utility function such that the Keras layers are able to work with the data
- Other parameters such as optimizers & learning rate schedulers may also cause slight differences.
- Once both models are trainable, we'll be able to create graphs and more thoroughly analyze differences between the two.

Future Work

Current Tasks

- Build a NNLearner individual in EMADE based on the VFL model and run standalone on it
- Create more robust testing procedures for our keras version of the torch models
- Further explore and implement image processing techniques to our dataset
- Extract truth value for Apical Translation of the spine in body regions
- Study models similar to the VFL model and incorporate changes

Sub Team Meeting:
6PM Thursday