Scoliosis Subteam Finals Presentation

Eric Chen, Nathan Zhong, Lucas Yim, Brian Zhang, Patrick Weng, Daniel You, Charlotte Hettrich, Ruchi Patel, Aditya Chandaliya, Yahya Hassan, Minseung Jung, Pujith Veeravelli, Pranav Malireddy, Brandon Conner, Adil Shaik, Aditya Chandaliya, Tariq Almawash, Aarav Shah

Overview

- Problem/Introduction
- Datasets
- Midterm Status
- 2nd Half work
- Current blocks/Future Work

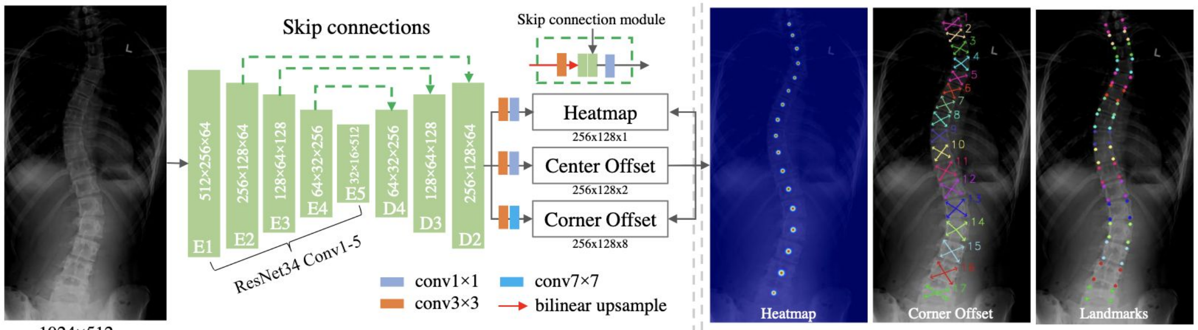
Problem

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.
- Different ranges determine different treatment methods

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.



1024×512 (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

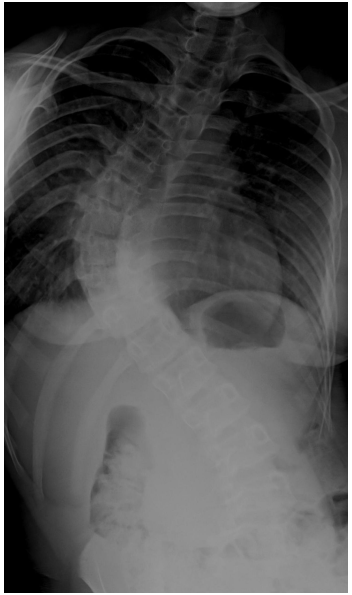
$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2.$$

Azure 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

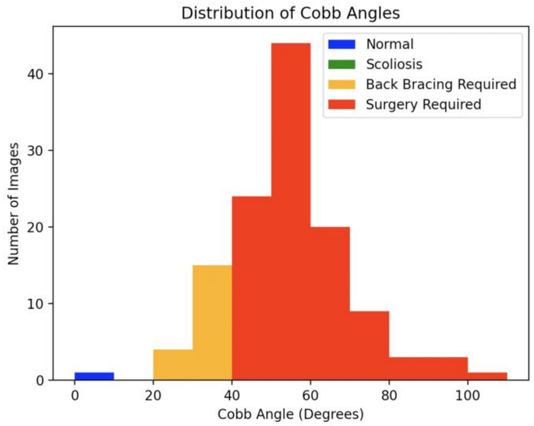
Datasets

Examples



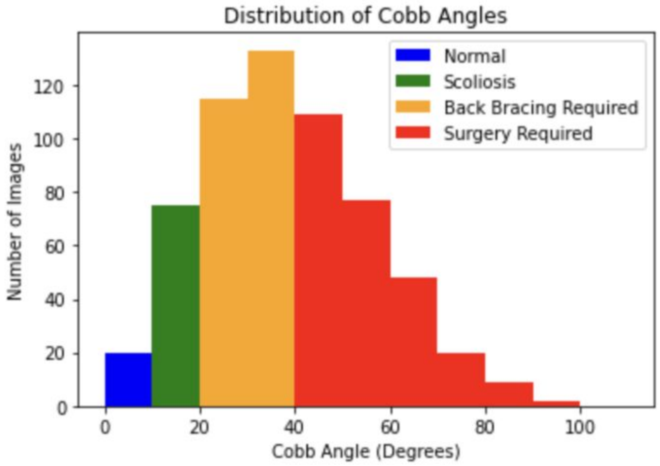
AASCE Dataset

- From AASCE 2019 Challenge for estimating spinal curvature
- 580 x-ray images with corresponding truth values for cobb angle and corner landmarks
- Landmark truth data provided by clinicians
- Challenge participants were evaluated on SMAPE



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
  - Adjusted the landmarks so the points were still formatted for the uncropped images
  - SMAPE Pre-Cropping: 16.5 -> SMAPE Post-Cropping 11.72
- Packaging to .npzs
  - Changed truth data to landmarks
  - Resampled the images and remapped truth data to consistent size

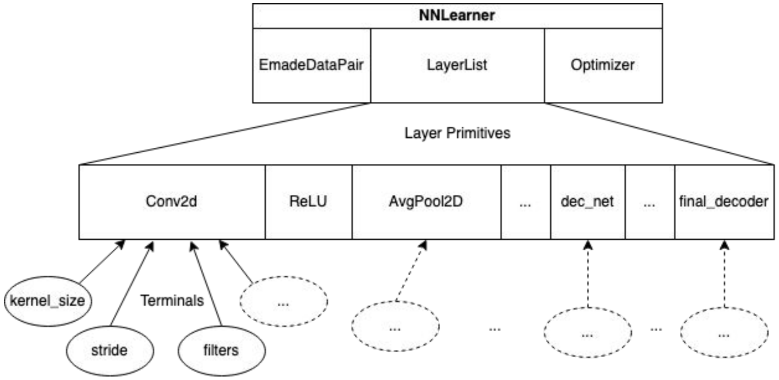
Midterm Status

Midterm Recap - Model Development and Conversions

- EMADE: Transitioned to NNLearner for model evolution, building models from ResBlocks and layers
- ResNet: Generic feed-forward layers with skip connections
- DecNet: Three combination modules, common Keras layers, and two custom layers (CustomImageResizing, CustomDecNetLayer)
- Decoder: Converts output into landmark form, implemented in EMADE as a custom layer
- PyTorch to Keras conversion: 4 PyTorch layers in VFL Model, Keras equivalents with minor differences
- Testing script: Compares Keras and PyTorch models, output differences (~1e-5)

NNLearner

- EMADE evolution: NNLearner introduced for evolving complex models
- Builds models using ResBlocks and individual layers for more flexibility and customization
- Common format for adding layers/parameters to layerlist in EMADE with Azure
- Aims for full EMADE run with functioning NNLearner primitive to develop optimal models

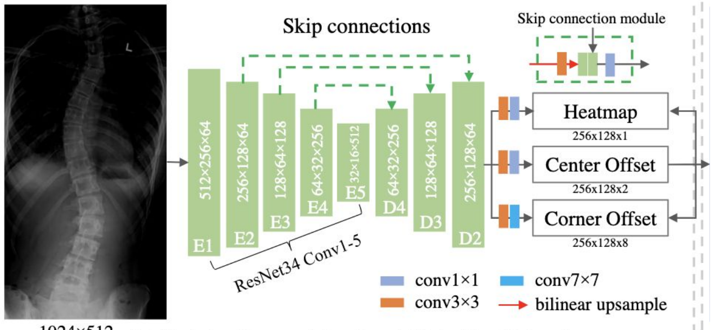



```
ARG0 = self.image_data
llist = nnm.InputLayer()
llist = nnm.ZeroPadding2DLayer(3, llist)
llist = nnm.OutputLayer(llist)
result = nnm.NNLearner(ARG0, llist,'adam')
```

```
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
_________________________________________________________________
```

Model Layers (Resnet/DecNet/Decoder)

- ResNet: Feed-forward layers with skip connections for efficient learning and reduced overfitting
- DecNet: 3 combination modules, common Keras layers, 2 custom layers (CustomImageResizing: resizes image based on provided dimensions, CustomDecNetLayer: outputs dictionary of heatmap, wh, and reg)
- Decoder: Converts output dictionary to landmark form (17 sets of 4 corner points), implemented in EMADE as custom layer (non-trainable weights) for pipeline evaluation



(a) Vertebra-Focused Landmark Detection Network

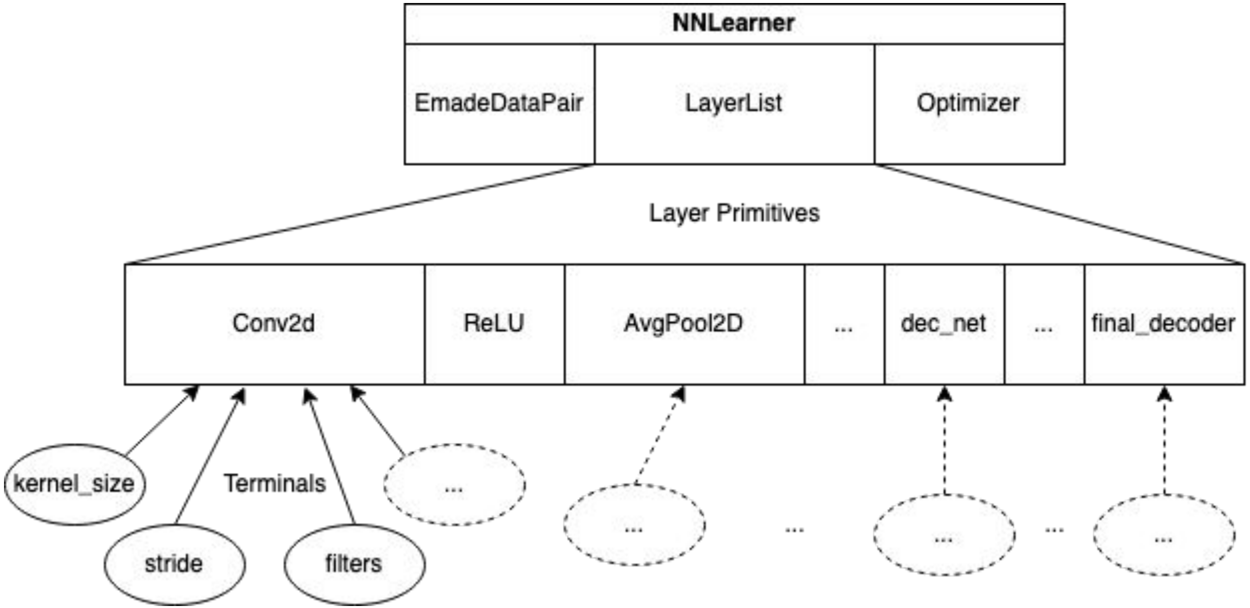
PyTorch to Keras Conversions

- VFL Model: 4 PyTorch layers converted to Keras equivalents with minor differences (padding adjustments using ZeroPadding2D, specifying data_format as "channels-first")
- Testing script: Compares Keras and PyTorch models using random input arrays, checks small output differences (~1e-5) to ensure accurate conversions
- Future plans: Build entire VFL model in Keras, convert training script dependencies (data loading, optimizers, learning rate schedulers), analyze differences in loss and performance once both models are trainable

```
# 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)
)
```

Primitives

Individual Structure



EMADE Primitives

- Layers
    - ResBlocks
    - Pooling
    - Convolutions
- ReLU
- Batch Normalization
- DecNet
- Decoder

```
#DecNet primitive
pset.addPrimitive(nnh.CustomDecNetLayer,[], nnm.LayerList, name="DecNet")

#Decoder primitive
pset.addPrimitive(nnh.VFLModelDecoder, [], nnm.LayerList, name="Decoder")
```

Work since Midterms

Preprocessing

Color Inversion:

- Certain images in Shriner's have had inverted colors (white background with black bones)
- Caused AASCE model to perform very poorly
- Used simple strategy of (255 - pixel) to invert images
- New fully preprocessing Shriner's image set located in

`nzhong31/Vertebra-Landmark-Detection-changed/crop_constrast_shriners_full`

With Inverted Images Removed
MSE of Landmarks: 342.36
SMAPE: 11.72

With Inverted Images
MSE of Landmarks: 345.30
SMAPE: 11.71

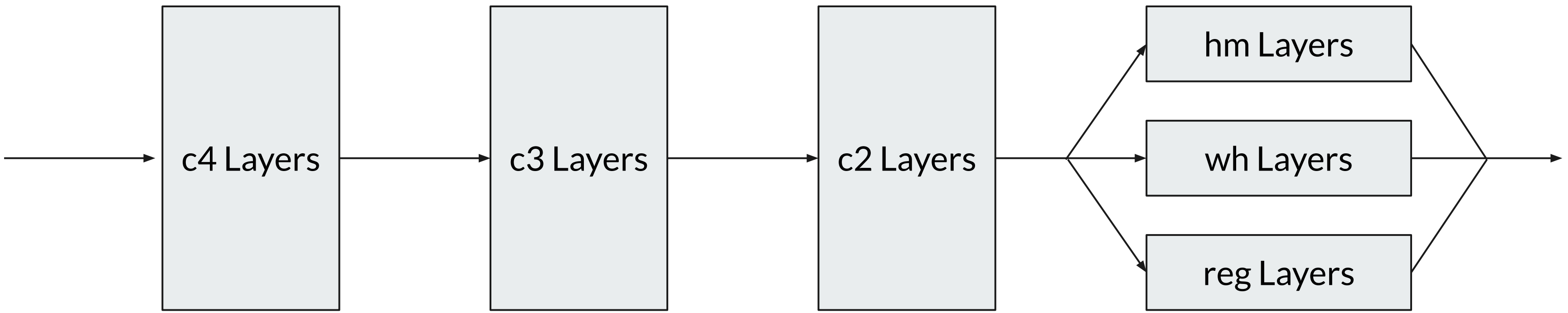
Model Architectures

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

DecNet

- Second part of the pipeline that takes ResNet output and outputs heatmap, wh, and reg
- Essentially reduces channel sizes of image progressively while incorporating “skip connections” from past layers
- Implemented as `CustomDecNetLayer` within `neural_network_helper.py`



Final Decoder

- Final part of the pipeline that takes in heatmap, wh, and reg, and turns it into the 68 landmarks
- Mostly mathematical calculations and array slicing/indexing
- Only change from original AASCE model -> Included padding at the beginning
    - Tied to the graph mode errors we were running into during compilation, as the input size during compilation doesn't align with the input size during runtime
    - Implemented "padding" at the beginning of the decoder that will help with dimensionality during compilation, but during runtime, size of tensors should match so no padding with occur
- Fully implemented in EMADE as `VFLModelDecoder` under `neural_network_helper.py`

NNLearner

Constructing our version of NNLearner

- When an individual creates and declares its NNLearner layers or blocks, a tuple is added to the layer list for that individual with the layer as a string as the first element

```
layerlist.mylist.append(('zeropadding2d', padding))
```

- Then when we compile the layer list, we iterate over the layer list and check what type of layer it is by indexing the first element
  - We then apply the corresponding Keras layer with correct parameters to the result of the previous layer and then continue with the rest of the layer list
- **Final Version as of Today:**
  - Changed from a layer granularity to a ResBlock granularity
  - NNLearner version of the AASCE model compiles and trains

Blockages

- Graph Mode Issues
    - EMADE NNLearner runs in graph mode which doesn't keep values for Tensors during compile time
    - Therefore, running `.numpy()` was an issue since no known values were present (Used this for slicing or getting shapes)
    - Running NNLearner with eager execution -> Placeholder error, which might be generated from the layerlist
    - Had to use workarounds to replace `.numpy()` calls in custom layers
- ResNet Dimensionality Issues
    - ResNet output shapes were too small as compared to what was expected from PyTorch model
    - Reason -> `BasicBlocks` in PyTorch model had an additional add call in the forward function that we didn't account for
- Changes to NNLearner
    - Changed `batch_size` from 100 to 1 (Ran into an `OutOfMemoryError`)
    - Removed automatic addition of Output layer (We have our own custom output layer)

BasicBlock Architecture:

```
def forward(self, x):
    identity = x

    out = self.conv1(x)
    out = self.bn1(out)
    out = self.relu(out)

    out = self.conv2(out)
    out = self.bn2(out)

    if self.downsample is not None:
        identity = self.downsample(x)

    out += identity
    out = self.relu(out)

    return out
```

Final Layers of the Model and their Outputs:

```
tf.nn.relu_31 (TFOpLambda)      (None, 16, 32, 512)  0          add_15[0][0]
________________________________________________________________________________
custom_dec_net_layer (CustomDec {'hm': (1, 2, 4, 1), 2924427    tf.nn.relu_31[0][0]
________________________________________________________________________________
vfl_model_decoder (VFLModelDeco (1, 68, 2)           0          custom_dec_net_layer[0][0]
                                                                custom_dec_net_layer[0][1]
                                                                custom_dec_net_layer[0][2]
================================================================================
```

Results of 5 Epochs of Training

```
Epoch 1/5
^[[A
481/481 - 273s - loss: nan - accuracy: 0.9097 - val_loss: 3498.0322 - val_accuracy: 0.9097
Epoch 2/5
481/481 - 227s - loss: 3498.0310 - accuracy: 0.9097 - val_loss: 3498.0322 - val_accuracy: 0.9097
Epoch 3/5
481/481 - 226s - loss: 3498.0317 - accuracy: 0.9097 - val_loss: 3498.0322 - val_accuracy: 0.9097
Epoch 4/5
481/481 - 227s - loss: 3498.0320 - accuracy: 0.9097 - val_loss: 3498.0322 - val_accuracy: 0.9097
Epoch 5/5
481/481 - 226s - loss: 3498.0322 - accuracy: 0.9097 - val_loss: 3498.0322 - val_accuracy: 0.9097
```

SpineNet() PyTorch-Keras Layer Comparison

SpineNet(
    ResNet-34(
        Max-Pooling

        Layer1: 3x Basic Block
        Layer2: 4x Basic Block
        Layer3: 6x Basic Block
        Layer4: 3x Basic Block)
    DecNet(
        C-4: 512 -> 256
        C-3: 256 -> 128
        C-2: 128 -> 64)
    Decoder(
        Hm: 256x128x1
        Reg: 256x128x2
        Wh: 256x128x8))

- Built full SpineNet model in PyTorch and Keras
- Compared model dimensions and node counts with model summaries
- Tested individual layer weight and bias conversion with test_torch_keras_conversion function

```
KERAS INPUT SHAPE: (1, 64, 64, 3)
TORCH INPUT SHAPE: torch.Size([1, 3, 64, 64])
1/1 [==============================] - 0s 275ms/step
Error: 0.0 is less than 1-e5? True
```

EMADE

Running EMADE

- Azure SQL database and all Azure infrastructure is still functional from last semester.
- Changes to Azure security complicates database access
    - Instead created notebook for visualization: visualize_database.ipynb
- EMADE template file in templates/input_scoliosis.xml
- Currently debugging our primitives and making sure our Pset is functional with EMADE

```
----------- INDIVIDUAL 0 ------------
hash : 0025e64ff62c8c799294b3caf4215c54c1ba552d38f247adf9ac7dc3aef29e82
elapsed_time : 0.003090381622314453
retry_time : 0.0
age : 0.0
evaluation_status : EVALUATED
evaluation_gen : 0
evaluation_start_time : 2022-12-10 00:45:38.650000
tree : cobb_prediction(spine_xray_cropping(ifThenElseDataPair(trueBool, ARG0, ARG0), passTriState(TriState.STREAM_TO_FEATU
RES), passAxis(Axis.AXIS_2), greaterThanEqual(-2.0790443534393797, 0.1)), passTriState(passTriState(TriState.FEATURES_TO_F
EATURES)), passAxis(passAxis(Axis.AXIS_0)), passFloat(myIntToFloat(2)))
```

Testing Scripts

PyTorch to Keras Conversion

- Wanted to recreate and train the model in Keras to compare it with the original PyTorch model

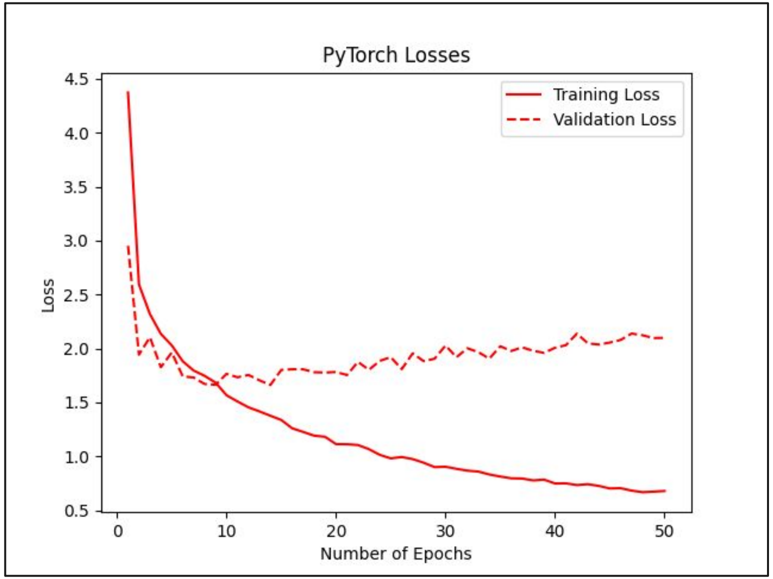
- VFL Model used a lot of custom scripts for training, all of which had to be converted to Keras:
  - models: Folder where SpineNet's code is stored
  - dataset.py: Inherits from PyTorch's DataLoader (used for reading images & creating batches)
  - loss.py: Inherits from PyTorch's Loss class; implements FocalLoss (for hm) and RegL1Loss (for wh, reg)
  - train.py: Custom training loop that sets up the model's optimizer (Adam) and scheduler (ExponentialLR) and runs it through epochs, manually updating the scheduler every loop

PyTorch vs. Keras Loss Metrics

- FocalLoss is used for “hm” (heatmap).
  - Focal Loss (FL) is an improved version of Cross-Entropy Loss (CE) that tries to handle the class imbalance problem by assigning more weights to hard or easily misclassified examples.
- RegL1Loss is used for “wh” (landmarks) and “reg” (corner offset).
  - L1 is Lasso Regression (Least Absolute Shrinkage and Selection Operator) adds “absolute value of magnitude” of coefficient as penalty term to the loss function.
  - Lasso shrinks the less important features coefficient to zero, removing some features altogether.
  - Works well for feature selection because we have a huge number of features.

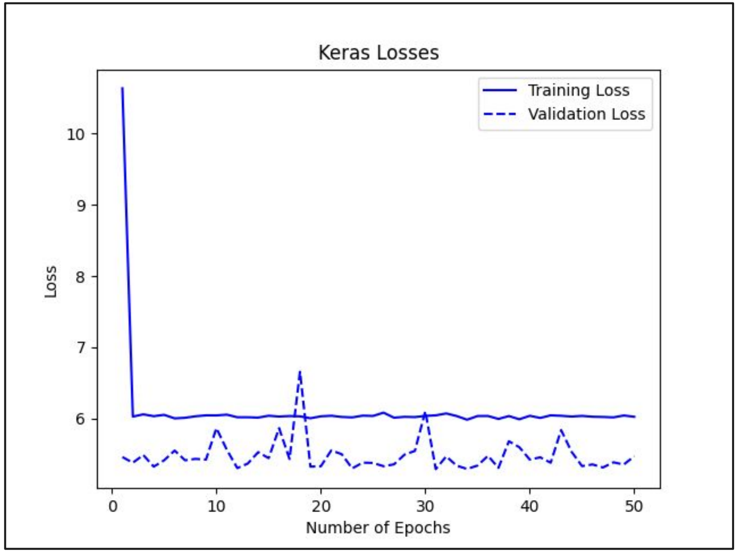
PyTorch Training and Validation Losses

- The PyTorch model is an unrepresentative validation of the dataset.
- Training process should be stopped when the validation error trend changes from descending to ascending.
  - If we stop the process before that point, the model will underfit.
  - If we stop the process after that point, the model will overfit.
- Could not evaluate the PyTorch model, this is a good future task.

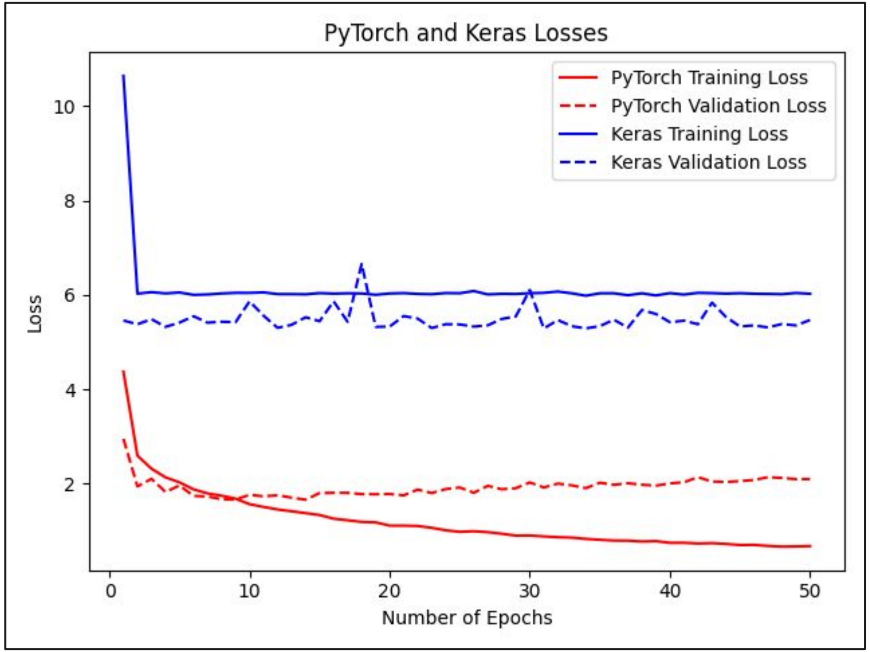


Keras Training and Validation Losses

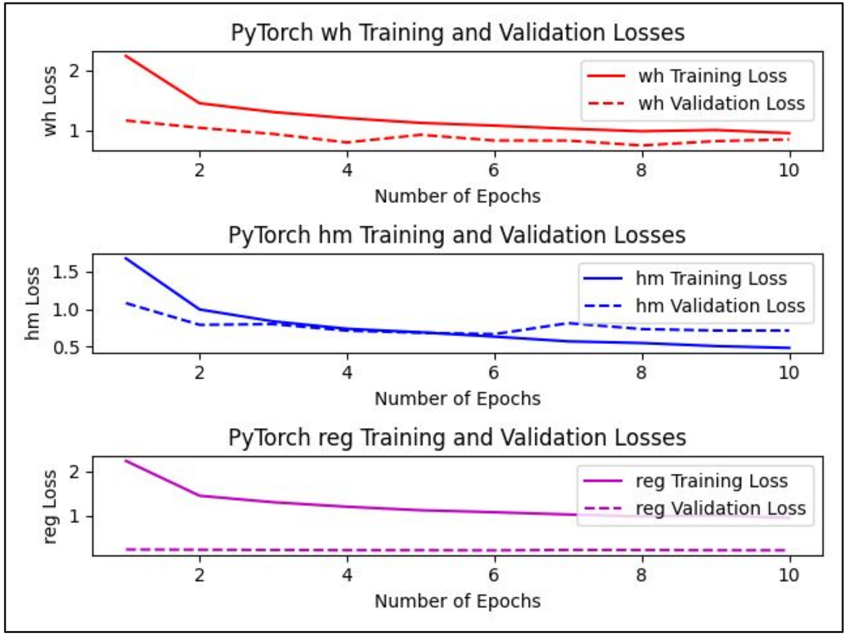
- The Keras model is an unrepresentative validation of the dataset.
  - Validation loss is lower than the training loss, therefore, the validation dataset is easier to predict than the training dataset.
- The training loss should continue to decrease and be similar in shape to an exp(-x) graph starting at origin.
- Values are much higher than the PyTorch model losses.

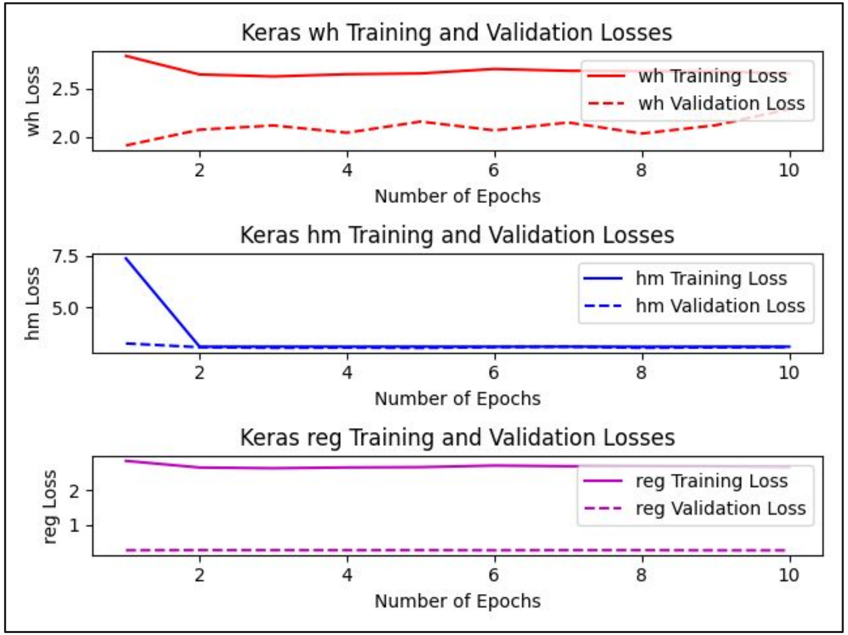


Comparison of PyTorch and Keras Losses

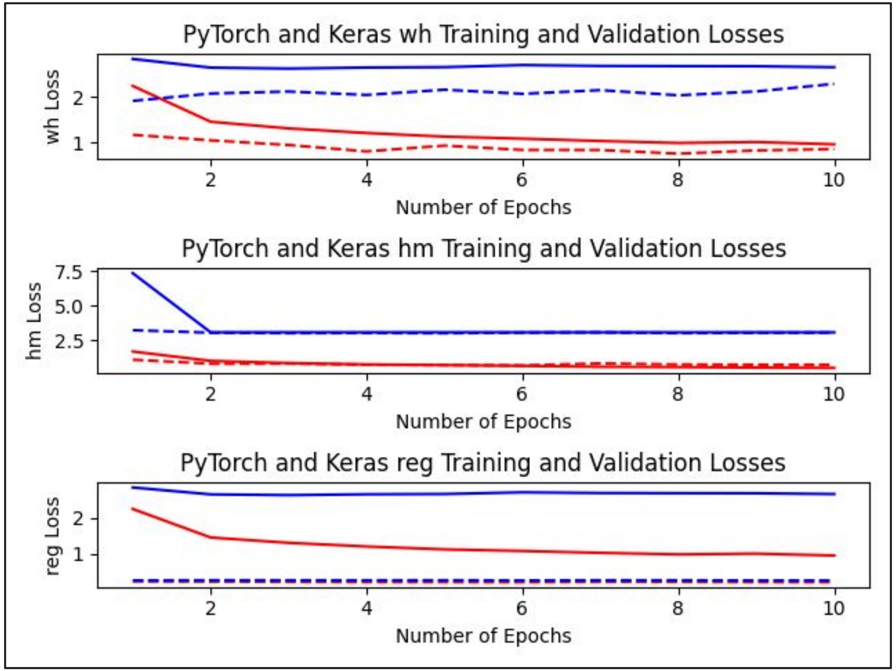


PyTorch and Keras Separated Losses





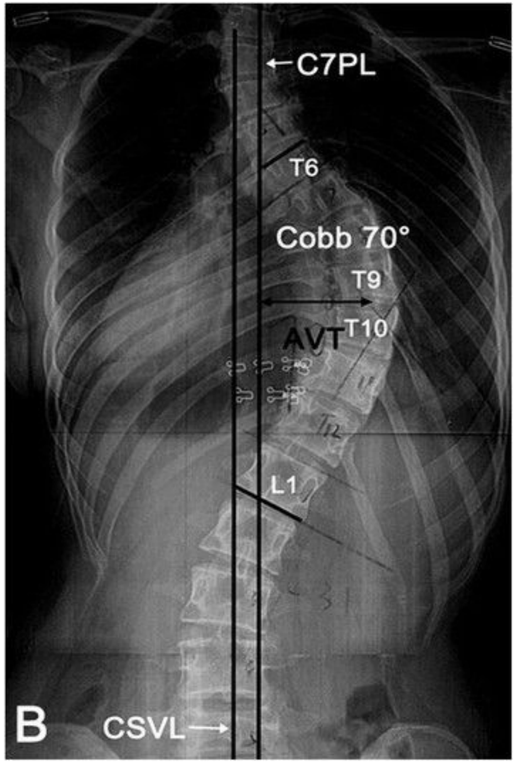
Combination of PyTorch and Keras Separated Losses



Apical Translation

Apical Vertebral Translation (AVT)

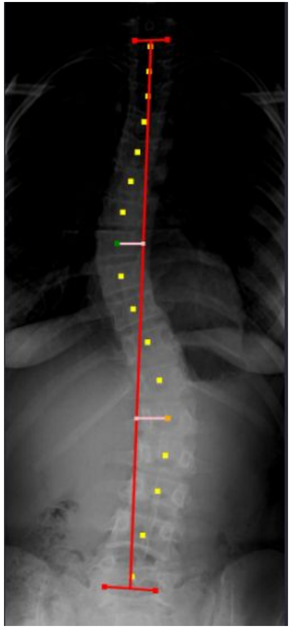
- Defined as the horizontal distance between the furthest (apical) vertebrae to the centerline of the spine
  - Thoracic apical translation uses the C7 plumb line
  - Lumbar apical translation uses the central sacral vertical line (CSVL)
- For initial calculations, used the line connecting centers of topmost and bottommost vertebrae
- Another useful metric to determine presence or degree of scoliosis
- Can extract truth data for it using the landmarks
  - Not much extra work



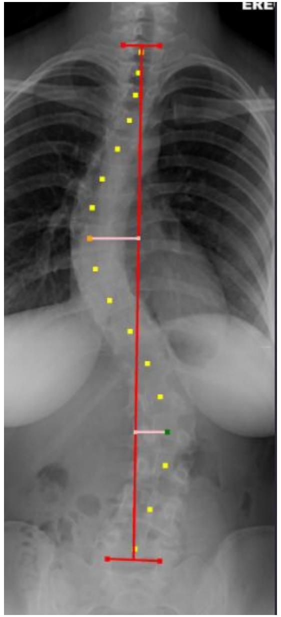
Results

- Accurately identifies AVT for some of the images
- Designed to recognize two vertebrae with the greatest distance, with the goal of distinguishing between thoracic and lumbar
- Distances still need to be converted to metric units (cm)

1st Distance: 99.375p
2nd Distance: 98.125p

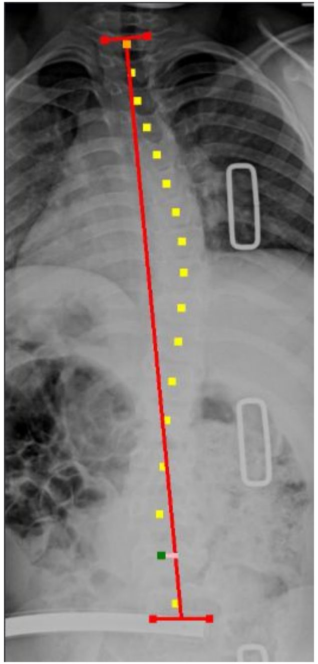


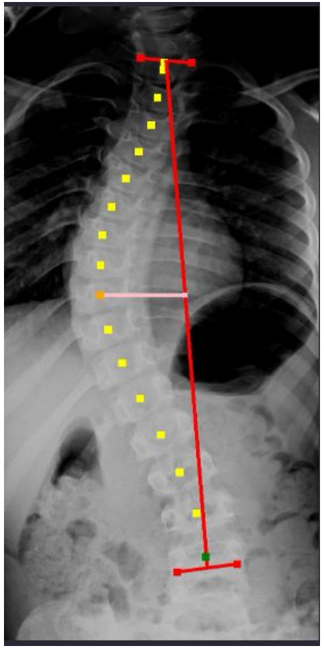
1st Distance: 190.625p
2nd Distance: 136.875p



Areas to improve

- For some of the images the AVT is shown very close to the centerline
  - Due to an issue with how landmarks are sorted, will be corrected soon
- In many images, there ends up being only one curve, due to combined scoliosis being a bit rarer
  - This often makes the second place point somewhat useless
  - Shown in the image on the right





Future Work

EMADE

Now that NNLearner is proven to be functional; we are ready to fully migrate everything to EMADE!

- Debug primitive registration process (errors with actually running EMADE)
- Increase the granularity of dec_net primitives; currently blocked together for testing NNLearner
- Determine where the loss discrepancies are coming from between Pytorch/Keras and reduce as much as possible
- Implement apical translation loss evaluation for EMADE
- “Lock” trainable parameters (resnet alone has millions of parameters)

Once these are finished; we are theoretically ready for EMADE runs