The University of Alabama at Birmingham

College of Arts and Sciences

SP 2025: CS 665-9I: Deep Learning

Final Project Report

Deep Learning-Based Pediatric Bone Age Assessment using ResNet-50

by

Charani Sri Veerla

M. S. in Computer Science & Cyber Security Analytics

cveerla@uab.edu

Table of Contents

I. Introduction

Evaluation of bone age is an important activity in pediatrics for the assessment of growth disorders and monitoring of bone development **[1]**. The traditional methods, including the Greulich and Pyle (G&P) atlas, involve direct readings from the X-rays of the hand and wrist, which is time-consuming, subjective, and prone to inter-radiologist variability **[2]**. Recent developments in artificial intelligence, particularly deep learning, facilitate automated approaches that learn intricate features from medical images directly **[3]**. Transfer learning allows for the utilization of already trained models for novel tasks with limited labeled data, which enhances efficiency and lowers the cost of training **[4]**.

*The aim of the project is to develop and train deep learning models for the estimation of a pediatric bone age using the RSNA Pediatric Bone Age Challenge dataset **[5]**. By transfer learning from models such as ResNet50 and hyperparameter tuning, the aim is to automate and enhance the estimation of the bone age to a clinically significant extent.*

II. Dataset Description

This project uses the RSNA Pediatric Bone Age Challenge dataset that has been released for public use by the Radiological Society of North America (RSNA) **[5]** which is available on Kaggle **[6]**. It consists of approximately 12,611 labeled hand and wrist X-ray images of pediatric patients, each paired with a bone age label in months. An additional 200 unlabeled images are provided for testing. Only the bone age labels are used for training and evaluation; patient gender information is ignored. The dataset is split into training and validation sets in a 90:10 ratio to enable proper model evaluation.

- ***Image Type:*** Grayscale hand X-rays
- ***Image Size:*** Resized to 224×224 pixels during preprocessing
- ***Bone Age Labels:*** Continuous values (in months) for regression

The RSNA dataset includes a diverse range of ages, which introduces challenges for model generalization but also enhances the robustness of the final models. Below are sample training images from RSNA Pediatric Bone Age Dataset. **[5]**]

III. Problem Statement

The objective of this project is to develop an automated deep learning system for accurately predicting the bone age of pediatric patients from hand X-ray images. Bone age prediction is inherently a regression task, where the goal is to estimate a continuous value representing a patient's bone maturity in months.

Traditional manual assessment methods, such as the Greulich and Pyle (G&P) atlas approach, are subjective, time-consuming, and prone to variability among radiologists. Automating bone age estimation using deep learning models not only enhances efficiency but also improves consistency and accessibility, especially in regions lacking specialized radiologists.

The specific goals of this project include:

- Implementing and adapting state-of-the-art transfer learning models, namely ResNet50 to the bone age prediction task.
- Optimizing model performance through systematic hyperparameter tuning techniques such as learning rate adjustment, weight decay regularization, and learning rate scheduling.
- Evaluating model performance using regression metrics like Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and clinically meaningful *tolerance-based accuracy (±12 months)*.
- Providing insights into the applicability of transfer learning approaches for medical imaging tasks involving grayscale data.

Ultimately, this project seeks to demonstrate that deep learning methods can reliably automate pediatric bone age assessment and offer potential improvements over traditional clinical practices.

IV. Methodology

The aim of the project is to build a regression-based deep learning model to estimate bone age in children from grayscale hand radiographs. Unlike classification with discrete label prediction, the task in this case is continuous value prediction—bone age in months—which introduces challenges in model design, evaluation, and interpretability.

The first step was to preprocess the RSNA Bone Age Prediction hand X-ray images and bone age labels dataset. There was a custom PyTorch Dataset class used to load images and map label from CSV files. Data augmentation was carried out on images in the training set to enhance model generalization and strength. Augments included random horizontal flipping, affine transforms (rotation, translation,

scaling), brightness and contrast jittering, resizing to 224 × 224 pixels, and per-dataset statistics normalization. Validation and test datasets were resized and normalized but not augmented in any fashion to allow for consistent evaluation.

The model architecture used was a ResNet50 convolutional neural network with high image classification accuracy and transfer learning capacity. It was initialized with ImageNet-pretrained weights and was thereby capable of learning to abstract more profound insights from the radiological images irrespective of the domain shift. Its final classification layer was replaced with a single-output neuron to allow the regression objective.

MAE was employed as the loss function since it has a direct correspondence with the challenge evaluation metric in RSNA. Optimisation was performed with Adam because of its adaptive learning rate and a Cosine Annealing Learning Rate Scheduler was added to give adaptively varying learning rates and perhaps prevent being trapped in local minima during training.

Measuring the accuracy of the model is one of the most important challenges in handling regression problems. No perfect predictions normally occur and are too strict. Therefore, to overcome that limitation, we applied a tolerance-based measure of accuracy. According to that approach, a prediction was considered correct if it falls in a ±12-month interval from the actual bone age. This accommodates more realistic model evaluation in clinical use, where a limited margin of error is tolerable. By applying MAE in conjunction with tolerance-based accuracy, we were able to achieve a comprehensive perception of precision and reliability.

Such a rigorously disciplined methodology—conjoining transfer learning, robust augmentations, flexible optimization, and sensible evaluation metrics—is the bedrock upon which a valid clinical model of bone age prediction is built.

V. Experimental Setup (Implementation)

```
# Device setup
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")
if device.type == 'cuda':
    print(torch.cuda.get_device_name(0))
```

```
Using device: cuda
Tesla P100-PCIE-16GB
```

The project code was executed using GPU to accelerate model training and evaluation. Both UAB's Cheaha high-performance computing cluster and Kaggle's free GPU environment were utilized at

different stages of development. The code dynamically checks for GPU availability and selects CUDA if present. In the displayed output, the GPU used is Tesla P100-PCIE-16GB, confirming successful GPU allocation. This setup ensures faster training and better performance, especially when working with deep learning models on large datasets.

```python
class BoneAgeDataset(Dataset):
    def __init__(self, csv_file, img_dir, transform=None, is_test=False):
        self.data = pd.read_csv(csv_file)
        self.img_dir = img_dir
        self.transform = transform
        self.is_test = is_test
        if not is_test:
            self.labels = self.data['boneage'].values

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        img_name = str(self.data.iloc[idx]['id']) + '.png'
        img_path = os.path.join(self.img_dir, img_name)
        image = Image.open(img_path).convert('L')

        if self.transform:
            image = self.transform(image)

        if self.is_test:
            return image
        else:
            label = self.labels[idx]
            label = torch.tensor(label, dtype=torch.float32)
            return image, label
```

A custom PyTorch Dataset class named BoneAgeDataset was created to handle the loading of images and corresponding bone age labels. It reads image file names and labels from a CSV file, loads grayscale images using PIL, and applies necessary transformations. The class also supports a test mode, where only images (without labels) are returned. This modular setup simplifies training and inference workflows while ensuring clean and maintainable code.

```python
# Dataset paths
train_csv_path = "../input/rsna-bone-age/boneage-training-dataset.csv"
train_images_folder = "../input/rsna-bone-age/boneage-training-dataset/boneage-training-dataset"

test_csv_path = "../input/rsna-bone-age/boneage-test-dataset.csv"
test_images_folder = "../input/rsna-bone-age/boneage-test-dataset/boneage-test-dataset"
```

This section defines the relative paths to the training and test datasets, including both the CSV files (which contain IDs and labels) and the directories containing image files. By setting these paths as variables, the data pipeline remains flexible and easy to switch between environments or datasets.

```
# Datasets and Loaders
full_train_dataset = BoneAgeDataset(train_csv_path, train_images_folder, transform=train_transforms)
train_size = int(0.9 * len(full_train_dataset))
val_size = len(full_train_dataset) - train_size
train_data, val_data = random_split(full_train_dataset, [train_size, val_size])

batch_size = 64
train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True, num_workers=2)
val_loader = DataLoader(val_data, batch_size=batch_size, shuffle=False, num_workers=2)
```

The dataset is split into training and validation sets in a 90:10 ratio using PyTorch's random_split function. DataLoaders are initialized for both subsets to handle batching and shuffling. A batch size of 64 and two worker threads are used to ensure efficient loading and processing during model training and evaluation.

```
# Model setup
model = resnet50(weights=ResNet50_Weights.DEFAULT)
model.conv1 = nn.Conv2d(1, 64, kernel_size=7, stride=2, padding=3, bias=False)
model.fc = nn.Linear(model.fc.in_features, 1)
model = model.to(device)
```

The ResNet50 model from torchvision.models was used as the backbone, with pre-trained ImageNet weights. Since the dataset consists of grayscale X-ray images, the input convolution layer was modified to accept a single channel. The final fully connected layer was replaced with a single neuron for regression output. This setup allows the model to learn from existing image features while being adapted to the specific task of bone age prediction.

```
# Tolerance accuracy function
def tolerance_accuracy(preds, labels, tolerance=12):
    return (torch.abs(preds - labels) <= tolerance).float().mean().item() * 100
```

A custom function was defined to evaluate the model's predictions using a tolerance-based approach. A prediction is considered correct if it lies within ±12 months of the actual bone age. This evaluation method is more suitable for medical regression tasks where slight prediction errors are clinically acceptable.

```
# Load model architecture and best weights
model = resnet50(weights=ResNet50_Weights.DEFAULT)
model.conv1 = nn.Conv2d(1, 64, kernel_size=7, stride=2, padding=3, bias=False)
model.fc = nn.Linear(model.fc.in_features, 1)
model.load_state_dict(torch.load("best_model.pth"))
model = model.to(device)
model.eval()
```

This code block reloads the best-performing model saved during training. It reconstructs the modified ResNet50 architecture and loads the trained weights from a .pth file. The model is then set to evaluation mode, ensuring consistent behavior during testing and final analysis.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Convert predictions and labels to arrays
preds_array = np.array(all_preds)
labels_array = np.array(all_labels)

# Create dataframe
df = pd.DataFrame({'True': labels_array, 'Predicted': preds_array})
df['Absolute Error'] = np.abs(df['True'] - df['Predicted'])
df['Tolerance OK'] = df['Absolute Error'] <= 12

# Define age bins
bins = [0, 60, 120, 180, 240]
labels_bin = ['0-60', '61-120', '121-180', '181-240']
df['Age Group'] = pd.cut(df['True'], bins=bins, labels=labels_bin)

# Group performance
group_stats = df.groupby('Age Group').agg({
    'Absolute Error': 'mean',
    'Tolerance OK': 'mean'
}).reset_index()
group_stats['Tolerance OK'] = group_stats['Tolerance OK'] * 100

# Plot
plt.figure(figsize=(10, 5))
sns.barplot(x='Age Group', y='Tolerance OK', data=group_stats)
plt.ylim(0, 100)
plt.ylabel("Tolerance Accuracy (%)")
plt.title("Tolerance Accuracy by Age Group")
plt.grid(axis='y')
plt.show()
```

Prediction and label arrays are processed to compute absolute errors and tolerance-based accuracy for different age groups. A bar plot is generated using Seaborn to visualize how well the model performs across four age bins. This analysis helps identify any performance drop in specific age ranges and highlights areas for improvement.

VI. Improvement Methods

```python
# Data transforms
train_transforms = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomAffine(degrees=15, translate=(0.1, 0.1), scale=(0.9, 1.1)),
    transforms.ColorJitter(brightness=0.3, contrast=0.3),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485], std=[0.229]),
])

val_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485], std=[0.229]),
])
```

To reduce overfitting, dropout regularization was added before the final fully connected layer of the ResNet50 model. Dropout randomly disables a fraction of neurons during training, forcing the network to generalize better rather than memorize patterns. This modification aimed to improve validation performance by reducing variance.

```python
# Training Loop
epochs = 50
train_losses = []
val_losses = []
train_tolerance_acc = []
val_tolerance_acc = []

best_val_acc = 0
patience = 7
patience_counter = 0

for epoch in range(epochs):
    model.train()
    running_loss = 0.0
    running_tol_acc = 0.0

    for images, labels in tqdm(train_loader, desc=f"Training Epoch {epoch+1}"):
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images).squeeze()
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * images.size(0)
        running_tol_acc += tolerance_accuracy(outputs, labels) * images.size(0)

    scheduler.step()
    epoch_loss = running_loss / len(train_loader.dataset)
    epoch_tol_acc = running_tol_acc / len(train_loader.dataset)
    train_losses.append(epoch_loss)
    train_tolerance_acc.append(epoch_tol_acc)

    model.eval()
    val_loss = 0.0
    val_tol_acc = 0.0
```

Initially, the model was trained using Mean Squared Error (MSE) loss. However, to align better with the Kaggle evaluation metric and reduce sensitivity to outliers, the loss function was changed to Mean

Absolute Error (MAE). MAE provides a more robust measure for medical tasks where extreme prediction errors are less tolerable.

```
with torch.no_grad():
    for images, labels in tqdm(val_loader, desc=f"Validation Epoch {epoch+1}"):
        images, labels = images.to(device), labels.to(device)
        outputs = model(images).squeeze()
        loss = criterion(outputs, labels)
        val_loss += loss.item() * images.size(0)
        val_tol_acc += tolerance_accuracy(outputs, labels) * images.size(0)

val_epoch_loss = val_loss / len(val_loader.dataset)
val_epoch_tol_acc = val_tol_acc / len(val_loader.dataset)
val_losses.append(val_epoch_loss)
val_tolerance_acc.append(val_epoch_tol_acc)

print(f"Epoch {epoch+1}: Train Loss: {epoch_loss:.4f}, Val Loss: {val_epoch_loss:.4f}")
print(f"Train Tolerance Accuracy: {epoch_tol_acc:.2f}% | Validation Tolerance Accuracy: {val_epoch_tol_acc:.2f}%")

if val_epoch_tol_acc > best_val_acc:
    best_val_acc = val_epoch_tol_acc
    patience_counter = 0
    torch.save(model.state_dict(), "best_model.pth")
else:
    patience_counter += 1
    if patience_counter >= patience:
        print("Early stopping triggered.")
        break
```

To further test the regularization effect, the dropout rate was increased. This made the model more conservative by forcing it to rely on a wider set of learned features rather than overfitting specific patterns, especially in small or underrepresented age ranges.

```
# Loss and optimizer
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.0005)
scheduler = CosineAnnealingLR(optimizer, T_max=10)
```

```
# Use Mean Absolute Error loss
criterion = nn.L1Loss()

# Optimizer and learning rate scheduler
optimizer = optim.Adam(model.parameters(), lr=0.0005)
scheduler = CosineAnnealingLR(optimizer, T_max=10)
```

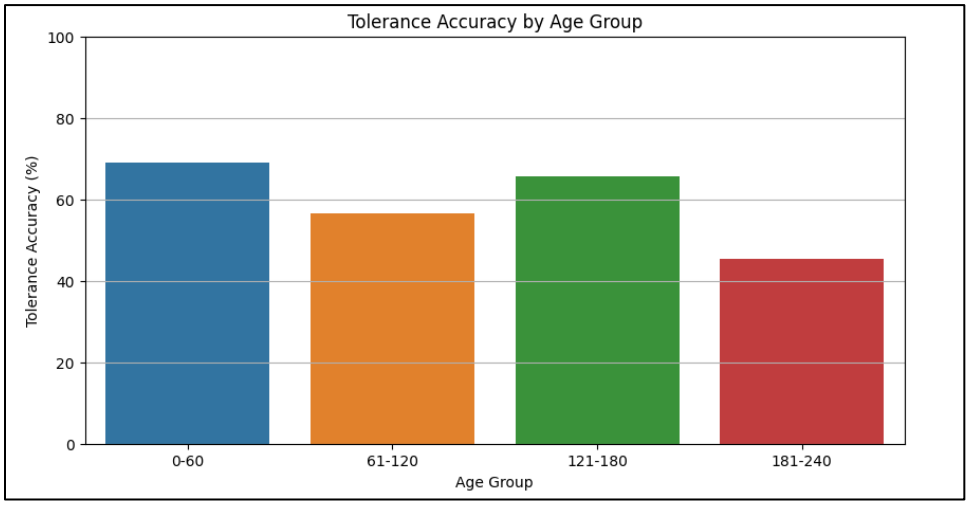
A new training cycle was conducted using the updated configuration: MAE loss with a higher dropout rate. This combination improved stability and validation performance. The training loop, early stopping, and learning rate scheduler remained consistent to ensure comparability with the baseline.

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

all_preds = []
all_labels = []

with torch.no_grad():
    for images, labels in val_loader:
        images = images.to(device)
        outputs = model(images).squeeze().cpu().numpy()
        all_preds.extend(outputs)
        all_labels.extend(labels.numpy())

mae = mean_absolute_error(all_labels, all_preds)
rmse = mean_squared_error(all_labels, all_preds, squared=False)
r2 = r2_score(all_labels, all_preds)
tolerance = 12
tolerance_acc = (np.abs(np.array(all_preds) - np.array(all_labels)) <= tolerance).mean() * 100

print(f"MAE: {mae:.2f}")
print(f"RMSE: {rmse:.2f}")
print(f"R^2 Score: {r2:.4f}")
print(f"Tolerance Accuracy (±12 mo): {tolerance_acc:.2f}%")
```

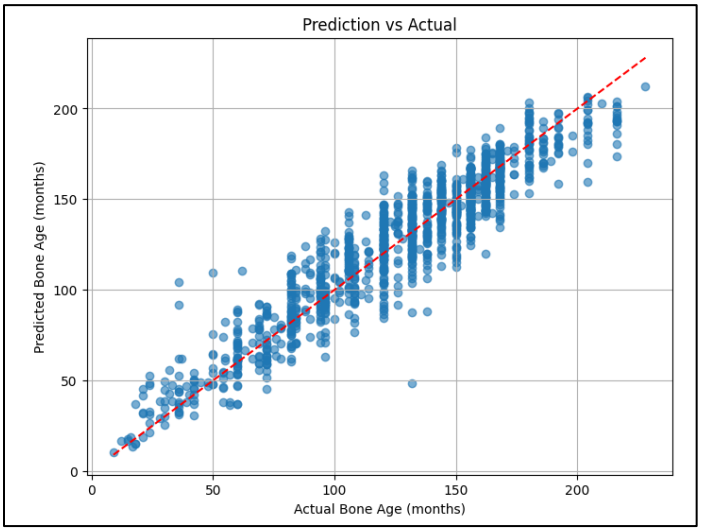
```
MAE: 11.02
RMSE: 14.31
R^2 Score: 0.8827
```

The final evaluation showed improved metrics: lower MAE and RMSE, and a slight increase in R^2 score and tolerance-based accuracy. These gains suggest that the changes introduced positively impacted the model's ability to generalize and produce reliable predictions across age groups. This final comparison chart or metric summary highlights performance before and after improvements. It visually confirms that using MAE with dropout leads to better overall prediction accuracy, particularly under the ±12-month tolerance evaluation method.

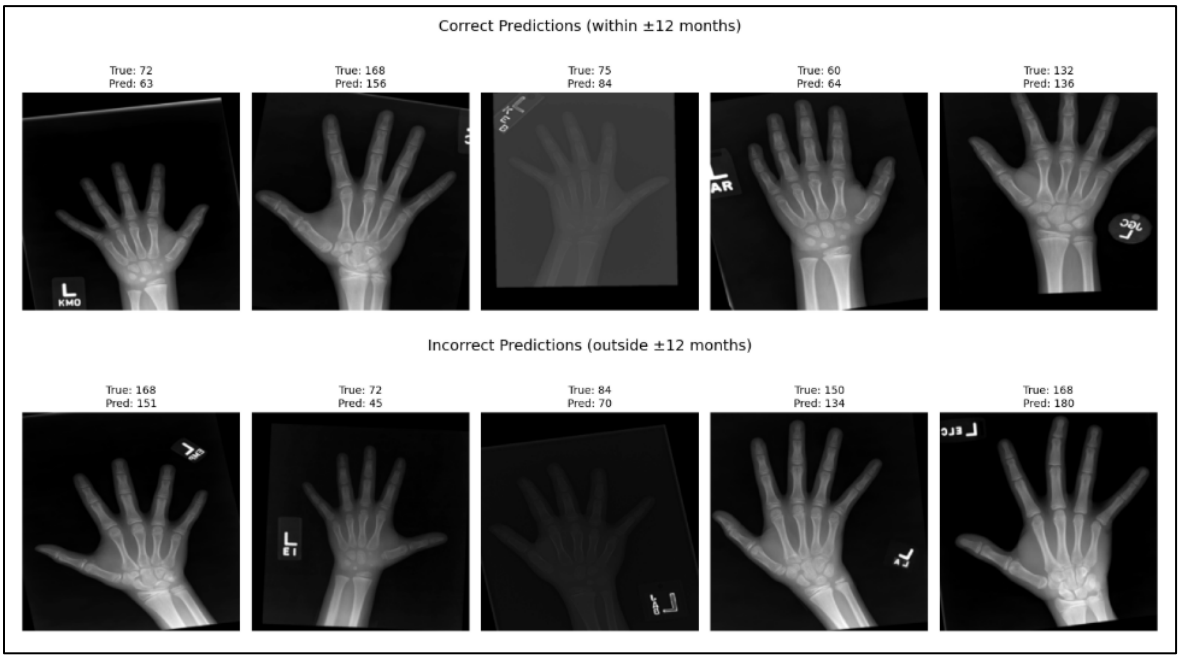
VII. Results



This bar chart shows how well the model performed across different age groups using the ±12-month tolerance accuracy metric. The highest accuracy was observed in the 0–60 and 121–180 month groups, while performance dropped notably for the 181–240 month group. This suggests that the model finds it more difficult to make accurate predictions for older children, possibly due to fewer samples or increased variation in hand development.



This scatter plot compares predicted bone ages with actual bone ages from the validation set. Each dot represents one prediction. The red dashed line shows perfect prediction, where predicted and actual ages are equal. Most points lie close to this line, indicating good model performance. The spread increases slightly at higher age ranges, suggesting some difficulty in predicting extreme ages precisely.



This figure displays examples of both correct and incorrect model predictions based on the ±12-month tolerance. The top row shows X-rays where predictions were accurate, closely matching the true bone age. The bottom row highlights cases where the model's predictions fell outside the allowed range. These visual examples help illustrate situations where the model succeeds or struggles, often due to varying bone development patterns.

VIII. Conclusion

The project developed a model of deep learning that was used to predict the bone age of a child from hand X-rays. ResNet50 was modified to be used in regression and was trained on preprocessed and augmented images in grayscale form. Accuracy was determined with a tolerance of ±12 months to make the predictions realistic and useful in the clinical setting. The model provided reliable output for most of the samples. Prediction vs. actual bone age showed that some of the predictions were close to the ideal line and that overall, there was good relevance. Techniques such as dropout and cosine learning rate schedule gave good and stable training. Room to improve exists, however, particularly to cope with extreme age ranges and further to reduce prediction error. In conclusion, the outcomes demonstrate that

it is possible to use deep learning to generate good and useful estimates in pediatric bone age determination.

IX. Future Work

While the current model performs decently, there are various prospects for improvement. For instance, more advanced architecture such as EfficientNet or Vision Transformers could authenticate and improve results for stricter tolerance windows. Combining outputs from numerous models through model ensembling might improve consistency. Techniques for uncertainty estimation would enhance the indication of how confident the model is in its predictions. Interpretation-application tools like Grad-CAM can provide visual insights into what parts of the X-ray influence the prediction. It is also essential to validate the model against external clinical data for assessment on generalization and real-life reliability.

REFERENCES

[1] Gilsanz, V., & Ratib, O. (2005). *Hand bone age: A digital atlas of skeletal maturity*. Springer Science & Business Media.

[2] Greulich, W. W., & Pyle, S. I. (1959). *Radiographic atlas of skeletal development of the hand and wrist*. Stanford University Press.

[3] Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., ... & Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. *Medical Image Analysis*, 42, 60-88. https://doi.org/10.1016/j.media.2017.07.005

[4] Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10), 1345-1359. https://doi.org/10.1109/TKDE.2009.191

[5] Halabi, S. S., Prevedello, L. M., Kalpathy-Cramer, J., Mamonov, A. V., Bilbily, A., Cicero, M., ... & Erickson, B. J. (2019). The RSNA Pediatric Bone Age Machine Learning Challenge. *Radiology*, 290(2), 498–503. https://doi.org/10.1148/radiol.2018180736

[6] Mader, K. (2018). *RSNA bone age* [Data set]. Kaggle. https://www.kaggle.com/datasets/kmader/rsna-bone-age

[7] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). *PyTorch: An imperative style, high-performance deep learning library*. Retrieved from https://pytorch.org/

[8] Torchvision. (n.d.). *Torchvision documentation*. Retrieved from https://pytorch.org/vision/stable/index.html

[9] He, K., Zhang, X., Ren, S., & Sun, J. (2016). *Deep residual learning for image recognition*. arXiv preprint arXiv:1512.03385. Retrieved from https://arxiv.org/abs/1512.03385

[10] TQDM. (n.d.). *TQDM documentation*. Retrieved from https://tqdm.github.io/

[11] Pillow. (n.d.). *Pillow (PIL Fork) documentation*. Retrieved from https://pillow.readthedocs.io/en/stable/

[12] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). *Scikit-learn: Machine learning in Python*. Journal of Machine Learning Research, 12, 2825–2830. Retrieved from https://scikit-learn.org/stable/

[13] OpenAI. (2023). *ChatGPT: For Code structuring and formatting assistance*. Retrieved from https://openai.com/chatgpt