THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Deep Learning
Final Project
October 15, 2024
Amir Jafari
Due: *Check the Syllabus*

Advanced Chess FEN Description Generation Using Vision Transformer(ViT)

Individual Project Report (Group 5)

DATS 6303_10_Deep_Learning

Fall 2024

Name: Bhanu Sai Praneeth Sarva

1. Introduction:

1.1 Project Overview

This Project explores the application of Deep nerural networks in predicting chessboard configurations represented as Forsyth–Edwards Notation (FEN) from visual data. By leveraging Vision Transformers (ViT), Alex net, Efficient net, and utilizing GPU-accelerated training, the system accurately interprets and processes chessboard images. The methodology involves partitioning images into grids, identifying each square's contents, and synthesizing these predictions into a cohesive FEN representation. This integration of Nerural networks with strategic games exemplifies the utility of deep learning in structured visual prediction tasks. The system addresses the inherent challenges of recognizing diverse chess piece configurations while preserving their spatial relationships—a critical aspect for generating valid FEN strings.

1.2 Outline of Shared Work:

1.2.1. Model Building: Vision Transformer (ViT) Implementation -

- I was responsible for implementing the ViT (Vision Transformer) model, which is a type of neural network used for image processing tasks. This model is particularly suited to tasks like chessboard analysis, where understanding the spatial relationships between elements (pieces) is crucial.
- Leveraged the pre-trained Vision Transformer (ViT) model from PyTorch's torchvision.models library, adapting it for the unique task of multi-class classification on chessboard grids.
- Enhanced the classification head to output 13 distinct classes representing chess pieces and empty squares, ensuring compatibility with FEN.

1.2.2 Model Training:

- Designed efficient training pipelines incorporating gradient scaling through PyTorch's Automatic Mixed Precision (AMP), enabling enhanced computational efficiency without compromising accuracy.
- Utilized the AdamW optimizer alongside a cosine annealing learning rate scheduler to stabilize and optimize the learning process, balancing convergence speed and precision.

1.2.3 Explainability : Utilising the Integrated Gradients:

- Implemented advanced explainability techniques using Captum's Integrated Gradients, illuminating the specific image regions that significantly influenced model predictions, thereby enhancing interpretability.

2. Model Building:

2.1 Rationale for Selecting Vision Transformer (ViT) :

The Vision Transformer (ViT) was chosen due to its advanced capability to process image data through a sequence-based approach, distinguishing it from traditional convolutional neural networks (CNNs). Unlike CNNs, which rely on local receptive fields and hierarchical feature extraction, ViT employs self-attention mechanisms that allow the model to capture global relationships across an image. This attribute is particularly advantageous for chessboard analysis, where the spatial arrangement of pieces plays a critical role.

ViT's patch-based processing treats each section of the image as a sequence token, enabling the model to evaluate positional dependencies effectively. This capability ensures that spatially interdependent features, such as the relationship between a king and its surrounding pawns, are recognized and leveraged for accurate prediction. Additionally, leveraging pre-trained ViT models reduces the computational resources required for training, as these models already possess robust general features learned from extensive datasets like ImageNet.

2.2 Theoretical Foundations :

ViT's architecture draws from the success of transformer models in natural language processing (NLP), adapting them for vision tasks. Each image is divided into fixed-size patches, linearly embedded, and processed through multi-headed self-attention layers. This allows the model to weigh the importance of various image regions dynamically, ensuring that critical features are emphasized during processing. The inclusion of positional encodings further enhances ViT's ability to consider spatial relationships explicitly.

Key reasons for selecting ViT include:

I. **Global Feature Recognition:** ViT's self-attention mechanism ensures the model captures both local and global relationships.

II. **Transfer Learning Benefits:** Pre-trained ViT models provide a strong starting point, minimizing the need for large domain-specific datasets.

III. **Scalability:** The architecture's patch-based processing and linear attention scaling make it well-suited for high-resolution image analysis.

2.3 Equations:

2.3.1 Patch Embedding:

Each image x is divided into non-overlapping patches:

$$x_p = [x_1, x_2, \ldots, x_N], \quad x_i \in \mathbb{R}^{P \times P \times C}$$

where $P \times P$ is the patch size, C is the number of channels, and $N=(H \times W)/P^2$ is the total number of patches for an image of size $H \times W$.

Each patch is linearly embedded:

$$z_0 = [x_p^1 W_e + b_e, x_p^2 W_e + b_e, \ldots, x_p^N W_e + b_e]$$

where W_e and b_e are the learnable weights and biases of the embedding layer.

2.3.2 Transformer Encoder :

Self –Attention Mechanism :

The attention mechanism is calculated as:

1. Query, Key, Value computation:

$$Q = z_i W_Q, \quad K = z_i W_K, \quad V = z_i W_V$$

2. Attention Scores :

$$A = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

where is the dimensionality of the key.

3. Weighted Sum :

$$z_i' = AV$$

Multi-Head Attention :

$$\text{MHA}(z_i) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h) W_O$$

Where :

$$\text{head}_i = \text{Softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i$$

W_O is the output projection matrix, and is the number of attention heads.

Layer Normalization and Residual Connection :

1. Residual Connection for Attention :

$$z_i^{(1)} = \text{LayerNorm}(z_i + \text{MHA}(z_i))$$

2. Residual Connection for Feed-Forward :

$$z_i^{(2)} = \text{LayerNorm}(z_i^{(1)} + \text{FFN}(z_i^{(1)}))$$

2.3.4 Feed –Forward Network(FFN) :

The feed-forward network is applied indipendently to each token :

$$\mathrm{FFN}(z_i) = \mathrm{GELU}(z_i W_1 + b_1) W_2 + b_2$$

Where :

- W1 and W2 : Weights of the two linear layers in the FFN.
- b1 and b2 : Biases of the corresponding layers.
- **GELU Activation Function :**

$$\mathrm{GELU}(x) = 0.5x \left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right)\right)$$

2.3.5 Final Classification :

Sofmax activation function is applied which helps the classification head to map the class token to the output representing the 13 possible classes (12 chess pieces and an empty square).

2.3.6 Integrated Gradients for Explainability

Purpose : Integrated Gradients is implemented as an explainability technique to provide insights into which parts of an input image most significantly influence the model's predictions for chess piece recognition and FEN generation. This method helps in interpreting and validating the model's decision-making process by:

1. Highlighting important features: It identifies and visualizes the regions of the input image that are most influential in the model's classification decision.

2. Providing interpretability: It offers a way to understand why the model made a particular prediction, which is crucial for building trust in the model's outputs.

3. Debugging model behavior: By revealing which parts of the image the model focuses on, it can help identify potential biases or unexpected behaviors in the model.

4. Validating learning: It confirms whether the model is paying attention to relevant parts of the chessboard and pieces, rather than irrelevant background features.

5. Improving model performance: Insights gained from Integrated Gradients can be used to refine the model architecture or training process to focus on more relevant features.

Configuration:

Baseline :

The baseline is set to a zero tensor with the same shape as the input image. This represents a neutral starting point from which attributions are calculated.

Steps :

The number of steps for approximating the integral is set to 50. This value balances computational cost with the accuracy of the approximation.

Target :

The target is set to the predicted class label for each grid, focusing the explanation on why the model made that specific prediction.

Mathematical Operation:

Integrated Gradients calculates attributions by approximating the following integral:

$$IntegratedGrads_i(x) = (x_i - x_i') \times \int_{\alpha=0}^{1} \frac{\partial F(x' + \alpha \times (x - x'))}{\partial x_i} d\alpha$$

Where :

- X is the input image
- X’ is the baseline (Zero tensor)
- F is the model’s Output for the target class
- i represents each pixel in the input image

This integral is approximated using a Riemann sum over the specified number of steps, providing a measure of each pixel's importance to the model's prediction.

The resulting attributions offer a pixel-wise explanation of the model's decision, highlighting which areas of the chessboard image were most influential in recognizing pieces and generating the FEN notation.

2.4 ViT Model Architecture:

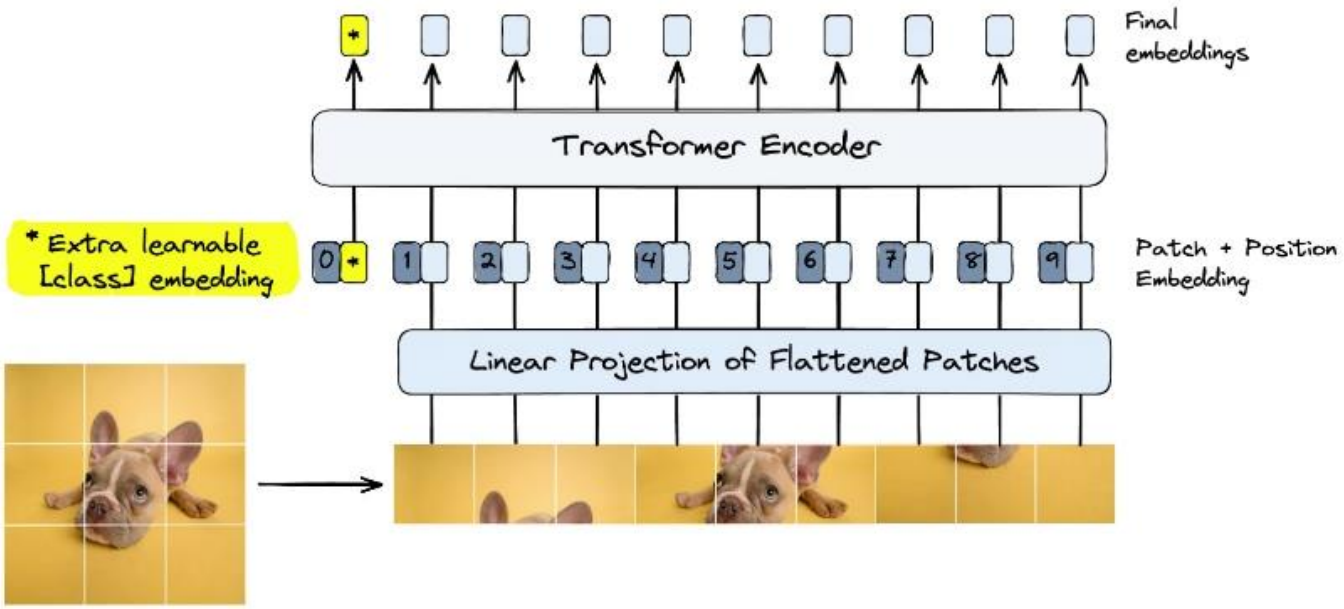


Figure 01. Model Architecture Layout.

```python
def get_model(pretrained: bool = True, num_classes: int = LABEL_SIZE) -> nn.Module:
    """
    Initialize and return the pre-trained Vision Transformer (ViT) model.
    """

    from torchvision.models import vit_b_16, ViT_B_16_Weights


    weights = ViT_B_16_Weights.IMAGENET1K_V1 if pretrained else None
    model = vit_b_16(weights=weights)


    for param in model.parameters():
        param.requires_grad = False


    num_ftrs = model.heads.head.in_features
    model.heads.head = nn.Linear(num_ftrs, num_classes)

    return model.to(DEVICE)
```

Figure 02. ViT model code

2.5 Integrated gradients :

I developed the explainability component using Captum's Integrated Gradients method to provide insights into the model's decision-making process. This helps in understanding which parts of the input chess images most significantly influence the model's predictions.

Key aspects of this implementation include :

1. Initializing the Integrated Gradients object :

```
ig = IntegratedGradients(model)
```

2. Defining a forward function for the specific prediciton :

```
def forward_func(x):
    outputs = model(x)
    probs = F.softmax(outputs, dim=1)
    return probs[:, pred_label]
```

3.Computing Attributions :

```
attributions, delta = ig.attribute(
    input_tensor,
    baselines=torch.zeros_like(input_tensor),
    target=pred_label,
    return_convergence_delta=True,
    n_steps=50
)
```

4. Visualizing the attributions :

```
attributions = attributions.squeeze(0).cpu().detach().numpy()
attributions = np.transpose(attributions, (1, 2, 0))
attributions = np.sum(np.abs(attributions), axis=2)
attributions = (attributions - np.min(attributions)) / (np.max(attributions) - np.min(attributions) + 1e-8)
```

5. Overlaying attributions on the original image :

```
overlay = original_image.copy()
overlay[:, :, 0] = overlay[:, :, 0] * 0.5 + attributions * 0.5
overlay[:, :, 1] = overlay[:, :, 1] * 0.5 + attributions * 0.5
overlay[:, :, 2] = overlay[:, :, 2] * 0.5
```

Figures : Integrated Gradients for Explainability code

This implementation allows for the generation of visual explanations for the model's predictions, highlighting the areas of the chess board that most influenced the FEN prediction.

These contributions significantly enhanced the project by improving training efficiency and providing valuable insights into the model's decision-making process, which is crucial for understanding and validating the chess FEN generation system.

3. Model Training:

One of the most important stages of the `Fen Generation Model` is the model training procedure. It entails configuring and carrying out a number of actions to use our dataset to optimise the model's parameters. The configuration, execution, and monitoring components of the training process carried out by the `train_model` function are covered in detail in this section.

3.1 Model Training Setup and Execution

3.1.1 Device Configuration: The training process is set up to utilize a GPU if available, leveraging its computational power for efficient processing of large datasets. If a GPU is not available, the model defaults to using the CPU.

3.1.2 Model Initialization: A pre-trained Vision Transformer (ViT) model is initialized and fine-tuned for the specific task of generating FEN strings from chessboard images. The model is moved to the chosen device to ensure consistent calculations.

3.1.3 Optimizer and Scheduler: The AdamW optimizer is used for parameter optimization, and a Cosine Annealing learning rate scheduler is employed to adjust the learning rate during training.

3.2 Training Process

3.2.1 Batch Processing: To optimize training efficiency, data is processed in batches. This approach significantly reduces memory consumption and allows for faster weight updates. By processing multiple samples simultaneously, the model can learn more effectively from the data.

3.2.2 Mixed Precision Training : To further accelerate training, mixed precision training is employed using PyTorch's torch.cuda.amp module. This technique involves performing computations with both 16-bit and 32-bit floating-point numbers. By leveraging the benefits of 16-bit precision for faster computations and the stability of 32-bit precision for critical operations, significant speed gains can be achieved without compromising accuracy.

```
scaler = torch.cuda.amp.GradScaler()
with torch.cuda.amp.autocast(enabled=True):
    outputs = model(inputs)
    loss = criterion(outputs, labels)
    _, preds = torch.max(outputs, 1)

if phase == 'train':
    scaler.scale(loss).backward()
    scaler.step(optimizer)
    scaler.update()
```

3.2.3 Performance monitoring : Throughout the training process, the model's performance is meticulously monitored. Key metrics such as loss and accuracy are tracked after each batch and at the end of each epoch.

- **Batch-Level Monitoring:** By examining these metrics after each batch, we can gain insights into the model's learning progress and identify potential issues early on.

- **Epoch-Level Review:** At the end of each epoch, a comprehensive evaluation of the model's performance on both the training and validation sets is conducted. This analysis helps assess the model's overall performance and generalization ability.

3.2.4 Model Saving : Once the training process is complete, the final model parameters are carefully saved to a file. This preserved model can be utilized for various purposes, including:

- **Further Evaluation:** The model can be loaded and evaluated on additional datasets to assess its performance on unseen data.

- **Continued Training:** The model can be further fine-tuned with additional data or different hyperparameters.

- **Deployment:** The saved model is deployed for stream lit.

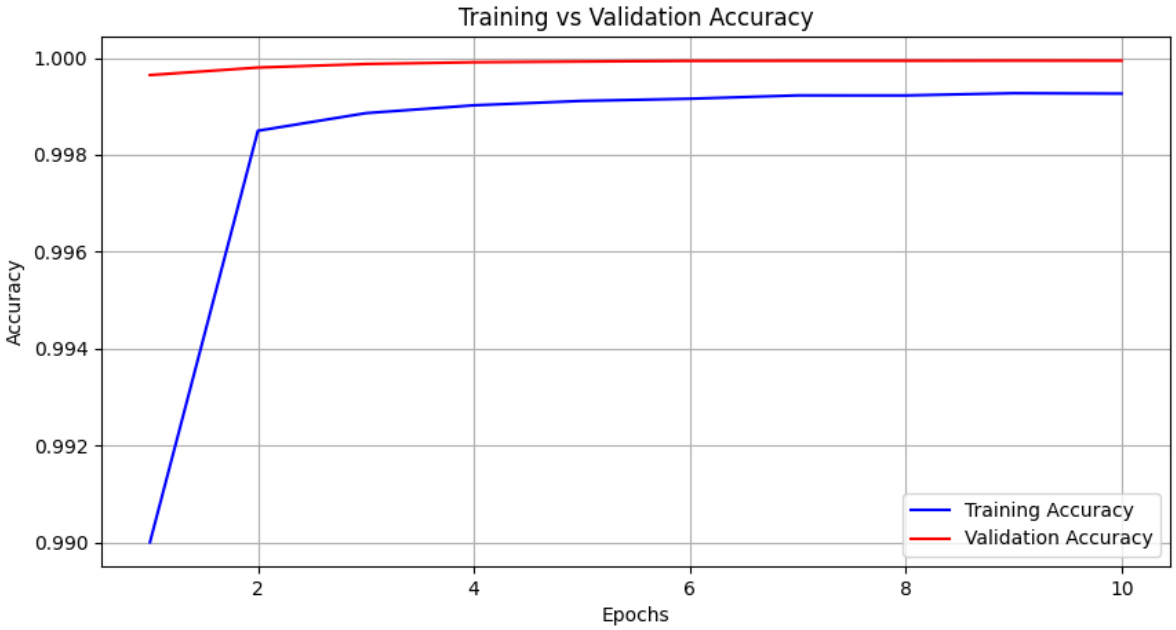
By incorporating these techniques, the model is trained efficiently and effectively, leading to improved performance and faster convergence.

4. Results :

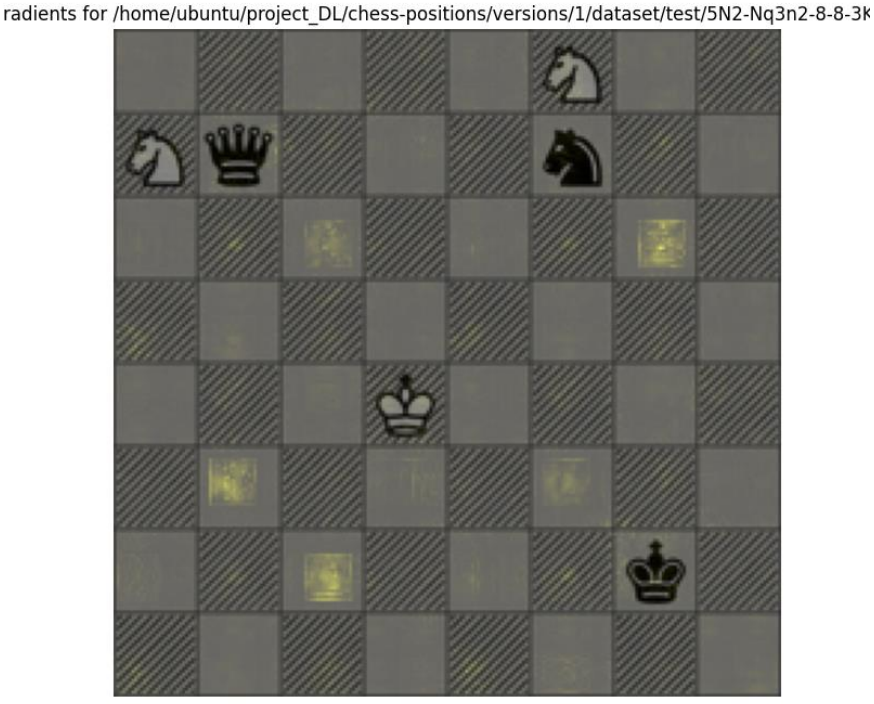
The model was trained for 10 epochs, achieving a final **validation loss of 0.0007**. The training and validation loss curves are shown below:

As observed, the model converges rapidly, with the validation loss decreasing steadily over the epochs.

The model's performance on the test set is impressive, with a **grid-level accuracy of 99.99%** and an **image-level accuracy of 99.55%** on the test set. This indicates that the model can accurately identify individual chessboard squares and generate correct FEN strings.



To gain insights into the model's decision-making process, Integrated Gradients (IG) was used to visualize the regions of the input image that contributed most to the model's predictions. These visualizations provide valuable information about the model's reasoning and help identify potential biases or limitations.



4.1 Future Work

- **Larger Dataset:** Training the model on a larger and more diverse dataset could further improve its performance and robustness.
- **Advanced Data Augmentation Techniques:** Exploring more sophisticated data augmentation techniques, such as style transfer and adversarial training, could enhance the model's generalization ability.
- **Ensemble Methods:** Combining multiple models, each trained on different subsets of the data or with different hyperparameters, could potentially improve overall performance.
- **Real-time Applications:** Integrating the model into real-time applications, such as chess analysis software or live streaming platforms, could provide valuable insights to players and analysts.

5. Conclusion

The developed Vision Transformer model, trained with mixed precision to accelerate the training process, has demonstrated impressive performance in generating FEN strings from chessboard images. The model's ability to accurately classify individual chessboard squares and generate correct FEN strings showcases its potential for various chess-related applications, such as automated game analysis, computer-aided chess instruction, and real-time chess position recognition.

To gain insights into the model's decision-making process, Integrated Gradients (IG) was employed to visualize the regions of the input image that contributed most to the model's predictions. This technique provides valuable information about the model's reasoning and helps identify potential biases or limitations.

6. Code Utilization:

Train file:

Original Lines from the internet: 58

Modified Lines: 6

Added line : 14

The percentage of the original code from the internet, after modifying and adding lines, is approximately 72.22%.

7. References:

- **Vision Transformer:** Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv preprint arXiv:2010.11929.

- **Integrated Gradients:** Sundararajan, M., Taly, A., & Yan, Q. (2017). Axiomatic attribution for deep networks. In International Conference on Machine Learning (pp. 3319-3328). PMLR.

Sources and related content

https://dl.acm.org/doi/10.5555/3305890.3306024