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 4)

DATS 6303_10_Deep_Learning
Fall 2024

Team members :
Bhanu Sai Praneeth Sarva
Ashwin Muthuraman
Bhoomika Nanjaraja
Sudhanshu Dotel

Introduction

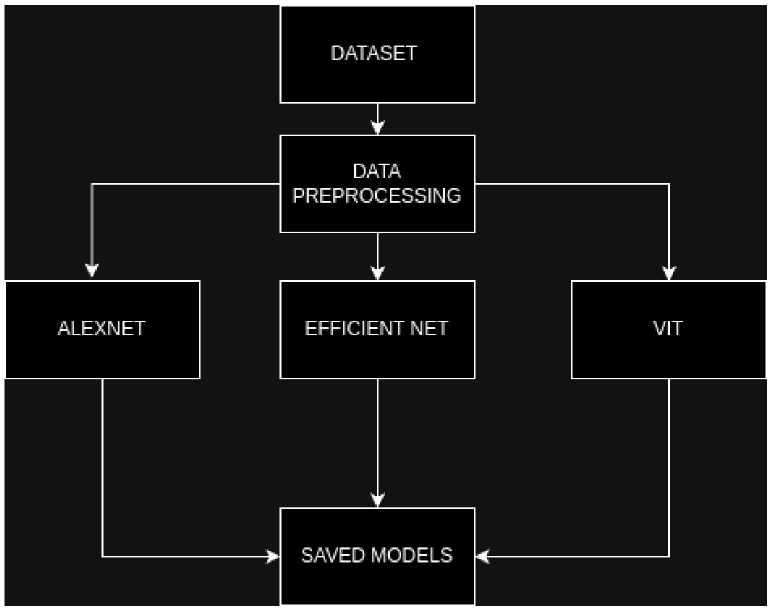
This project addresses a unique challenge in computer vision by implementing and comparing multiple state-of-the-art deep learning architectures for classification and object detection tasks. The aim was to evaluate the performance of Vision Transformer (ViT), AlexNet, and EfficientNet-B7 on the same dataset using identical preprocessing techniques. These models were trained and tested systematically to ensure a fair comparison of their strengths, limitations, and effectiveness in tackling image classification problems. Each architecture's implementation provided valuable insights into how diverse neural networks handle the same dataset and preprocessing methods.

In addition to classification tasks, the project extended to an advanced object detection pipeline using the YOLOv7 model. The goal was to train YOLOv7 to accurately detect chessboards in images, a specialized task requiring precision and robustness. To achieve this, a comprehensive preprocessing pipeline was developed to enhance the model's performance. Key preprocessing steps included adaptive thresholding to emphasize edges and regions of interest, contour detection to isolate key shapes, and selecting the largest contour to identify the primary object (the chessboard).

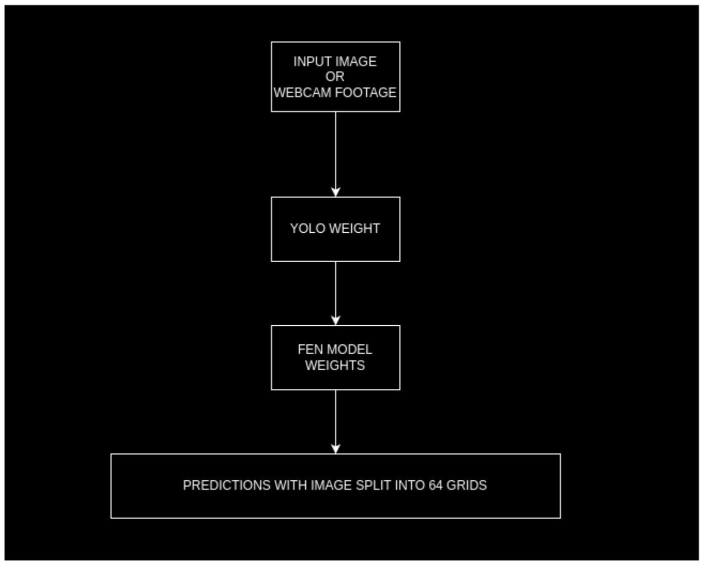
To refine the detection process, advanced techniques such as corner detection and perspective shifting were employed. Corner detection pinpointed critical points on the chessboard, while perspective shifting aligned and cropped the chessboard into a standard grid of 64 squares, ensuring consistency for downstream tasks. These methods not only improved detection accuracy but also streamlined the image preprocessing phase.

Finally, the project integrated all components into a user-friendly application using Streamlit. This deployment allowed users to upload images, detect chessboards, remove noise, and crop relevant regions effortlessly. By combining cutting-edge deep learning models with an intuitive interface, this project highlights the synergy between advanced machine learning techniques and real-world applications.

Project Flow :



Training flow


INPUT IMAGE OR WEBCAM FOOTAGE
YOLO WEIGHT
FEN MODEL WEIGHTS
PREDICTIONS WITH IMAGE SPLIT INTO 64 GRIDS

Stream lit flow

Chess board data fundamentals

- **Positioning**: The placement of pieces on the board according to the rules of chess, described using algebraic notation (e.g., e4, Nf3).

- **Fen (Forsyth-Edwards Notation)**: A standard notation used to describe a specific game state. It encodes the position of all pieces, whose turn it is, castling rights, en passant targets, and the move count.

- **Monochromatic/Polychromatic Squares**: Chessboards alternate in colors (light and dark squares). A **monochromatic square** refers to a single square's color, while a **polychromatic perspective** involves recognizing patterns and piece arrangements across multiple squares.

- **Piece Dynamics**:

  - **Static**: Pieces that remain unmoved or are strategically placed.
  - **Dynamic**: Actively moving pieces aiming for control, capture, or positioning.

- **Game Phases**:

  - **Opening**: Early moves aimed at piece development and control of the center.
  - **Middle Game**: The transition where strategies unfold, and tactical play begins.
  - **Endgame**: The phase where fewer pieces remain, focusing on precise calculations and pawn promotion.

- **Chessboard Grids**:

  - The board comprises 8x8 squares, forming a grid of 64 total spaces. Patterns such as ranks (horizontal rows) and files (vertical columns) structure gameplay.

Dataset Description

The dataset consists of 100,000 high-resolution images, each showcasing a randomly generated chess position. Each position contains 5 to 15 chess pieces, with the inclusion of 2 mandatory kings and 3 to 13 additional pieces, such as pawns, bishops, knights, rooks, and queens. These images are standardized at 400x400 pixels, providing sufficient clarity for visual processing while maintaining computational efficiency.

The images were created in kaggle using a custom-built generation tool that introduced significant diversity by employing 28 distinct chessboard styles and 32 different chess piece designs. This variability resulted in a total of 896 unique combinations of board and piece styles, ensuring that the dataset mimics real-world variability in chessboards and pieces.

To facilitate effective model training and evaluation, the dataset is split into 80,000 images for training and 20,000 images for testing, adhering to a conventional 80-20 train-test ratio. The placement of pieces in each image follows a predefined probability distribution: 30% of the pieces are pawns, while bishops, knights, and rooks each make up 20% of the additional pieces. Queens are comparatively rarer, constituting only 10%. Regardless of these distributions, two kings are always present on the board, as per the rules of chess.

Each image is labeled with the board position encoded in Forsyth–Edwards Notation (FEN), a standard format for describing chess states. However, instead of using traditional slashes to separate ranks, this dataset employs dashes as delimiters, adding a minor customization to the FEN format. This labeling ensures precise alignment between each image and its corresponding chess position, crucial for supervised learning tasks.

Data Preprocessing:

- **Image Splitting**:
    - The chessboard image is divided into 64 smaller grids (8x8), each corresponding to a single square on the board.
    - Ensures each grid is treated as an independent input for the model to classify pieces and their positions.
- **Image Resizing**:
    - Each grid is resized to 224×224 pixels to match the input size required by pre-trained models like AlexNet, VIT and ExceptionNet
    - Maintains compatibility with the model and reduces computation time.
- **Normalization**:
    - Images are normalized using the mean and standard deviation values typical for ImageNet pre-trained models.
    - Standardizes input data for faster convergence and better generalization.
- **Augmentations**:
    - Applied augmentations include rotation, random brightness/contrast, grid distortion, and elastic transform.
    - Prevents overfitting and improves the model's robustness to real-world variations like lighting changes or slight distortions.
- **FEN Mapping**:
    - A dictionary maps FEN characters (e.g., 'p', 'P', etc.) to numerical labels for model compatibility.
    - Includes all chess pieces (white/black) and an empty grid.

```
FEN_TO_LABEL_DICT = {
'p': 1, 'P': 2,
'b': 3, 'B': 4,
'r': 5, 'R': 6,
'n': 7, 'N': 8,
'q': 9, 'Q': 10,
'k': 11, 'K': 12,
'0': 0 # Representing empty grid
}

LABEL_TO_FEN_SHORT = {
0: "emptyGrid",
1: "p", 2: "P",
3: "b", 4: "B",
5: "r", 6: "R",
7: "n", 8: "N",
9: "q", 10: "Q",
11: "k", 12: "K"
}
```

- **Data Loading**:
  - Custom dataset class handles loading, preprocessing, and labeling.
  - Data loaders ensure efficient batching, shuffling, and parallel processing to optimize training performance.

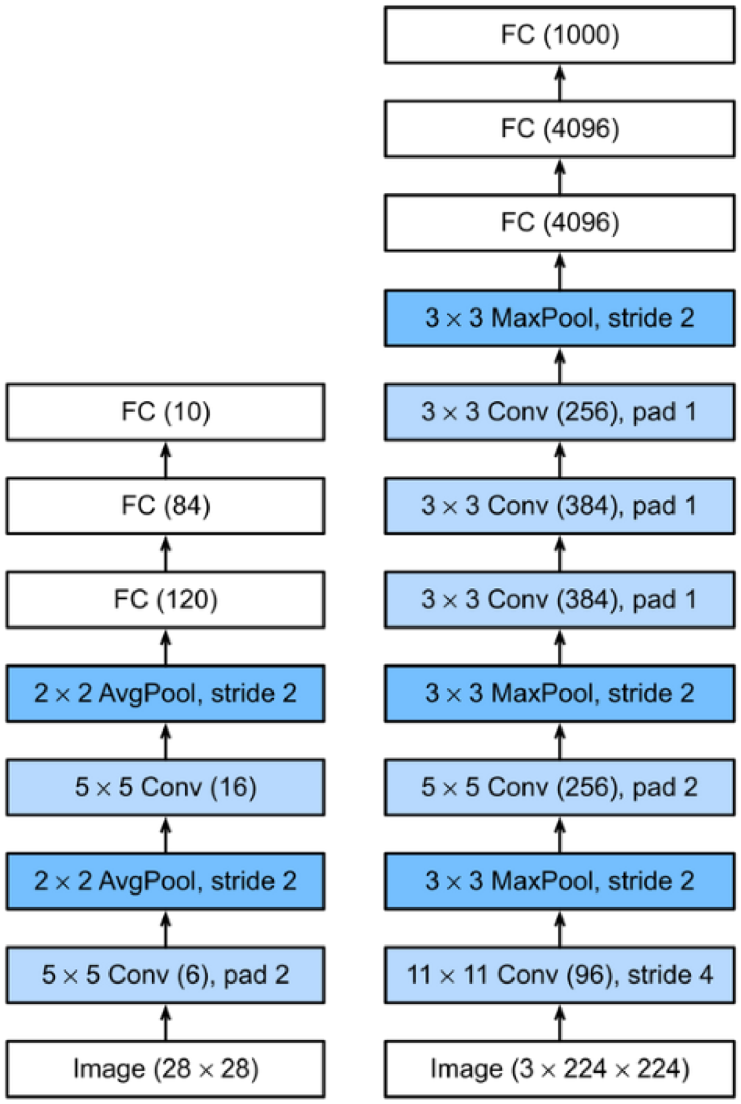
Deep learning network training and algorithms :

1. Alexnet :

Background Information

This project implements the **AlexNet architecture**, a foundational Convolutional Neural Network (CNN) for image classification. AlexNet, designed with a layered structure, incorporates convolutional layers for feature extraction, pooling for dimensionality reduction, and fully connected layers for classification. The goal of the model is to classify individual grid cells of a chessboard into one of 13 possible categories, including 6 types of white pieces, 6 types of black pieces, and an empty square.

Network architechture :



1. **Convolutional Layers**

   a. **Purpose**: Extract spatial features from the input image.

   b. **Details**: AlexNet utilizes multiple convolutional layers with varying kernel sizes to capture hierarchical features, from edges in early layers to complex patterns in later layers.

   c. **Equation**:

$$Y_{i,j,k} = \sum_{m,n} X_{(i+m)(j+n)} \cdot W_{mnk} + b_k$$

Where:

- X : Input grid image (50x50 or resized 224x224 pixels).
- W_mnk : Convolution kernel weights of size m×n.
- B_k : Bias term for the k-th feature map.

2. **Activation Function**

   a. **Purpose**: Introduce non-linearity.
   b. **Details**: Rectified Linear Unit (ReLU) is applied after each convolution to enable learning of complex features.
   c. **Equation**:

$$f(x) = \max(0, x)$$

3. **Pooling Layers**

   a. **Purpose**: Downsample feature maps to reduce computational complexity and overfitting.
   b. **Details**: Max pooling selects the maximum value within a sliding window across the feature map.
   c. **Equation**:

$$Y_{i,j} = \max_{m,n} X_{(i+m)(j+n)}$$

Where m x n defines the pooling window dimensions.(e.g., 2x2)

4. **Fully Connected Layers**

   a. **Purpose**: Aggregate extracted features for final classification.
   b. **Details**: AlexNet employs dense layers with ReLU activations to learn complex decision boundaries.
   c. Equation :

$$Y = W \cdot X + b$$

Where W and b are the weights and biases, and X is the flattened feature vector from the convolutional layers.

5. **Dropout**

   a. **Purpose**: Prevent overfitting by randomly deactivating neurons during training.

b. **Details**: Applied with probabilities p=0.5 in the fully connected layers.

6. **Dropout**

   a. **Purpose**: Regularize the model by randomly omitting neurons during training to prevent overfitting.
   b. **Details**: Applied after the fully connected layers.
   c. **Dropout Rate**: p = 0.5

Training Algorithm

1. **Initialization**

   a. **Strategy**: He initialization for weights, which is optimized for ReLU activations.
   b. **Equation**:

$$W \sim \mathcal{N}(0, \sqrt{\frac{2}{n_{in}}})$$

Where n_in is the number of input neurons.

2. **Optimization**

   a. **Algorithm**: Adam optimizer is used for adaptive learning rates, balancing momentum and RMSProp techniques.
   b. **Equation**:

$$\theta_{t+1} = \theta_t - \alpha \frac{m_t}{\sqrt{v_t} + \epsilon}$$

Where m_t and v_t are the first and second moment estimates, α is the learning rate, and ϵ ensures numerical stability.

3. **Loss Function**

   a. **Type**: Cross-entropy loss for multi-class classification.
   b. **Equation**:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c})$$

Where y_i,cis the true label and y_hat_i,c is the predicted probability for class c.

4. **Regularization**

    a. **Technique**: Dropout is applied within fully connected layers to improve generalization and reduce overfitting.
    b. **Dropout Probability**: p = 0.5.

5. **Batching**

    a. **Mini-Batches**: Training uses mini-batches of size 256 for efficient gradient updates.
    b. **Epochs**: Model is trained over 15 epochs with early stopping to prevent overfitting.

6. **Learning Rate Scheduler**

    a. **Type**: ReduceLROnPlateau, which decreases the learning rate if the validation loss plateaus.
    b. **Factor**: 0.5
    c. **Patience**: 2 epochs.

Evaluation

1. **Accuracy Metric**

    a. **Purpose**: Measure the fraction of correctly classified grid cells.
    b. **Equation**:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Predictions}}$$

2. **Precision, Recall, F1-Score**

    a. **Purpose**: Evaluate model performance for imbalanced datasets.
    b. **Equations**:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

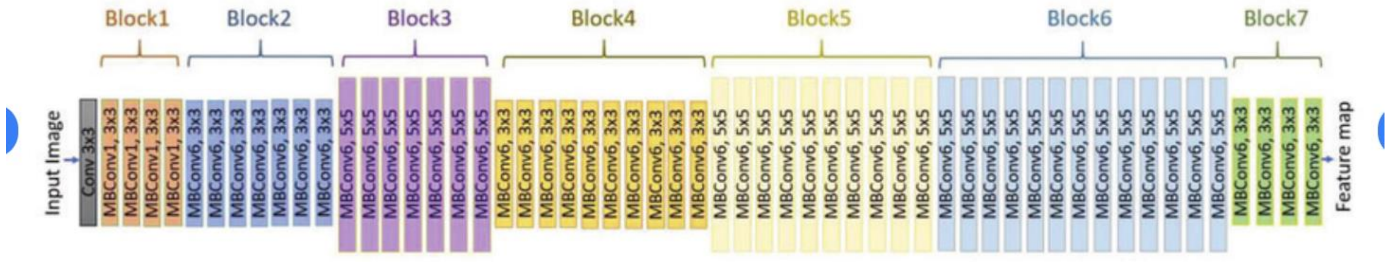
3. **Validation Set**: Used to monitor training progress and trigger early stopping if overfitting is detected.

4. **Test Set**: Final evaluation on unseen data to assess generalization.

This architecture and training pipeline ensure efficient feature extraction, robust learning, and accurate classification of chessboard grids into their respective categories.

2. EfficientNet_B7:

This project implements the EfficientNet architecture, a state-of-the-art Convolutional Neural Network (CNN) for image classification. EfficientNet, designed with a compound scaling strategy, incorporates depthwise separable convolutional layers for feature extraction, global average pooling for dimensionality reduction, and a fully connected classification layer. The model has been modified to classify individual grid cells of a chessboard into one of 13 categories, including 6 types of white pieces, 6 types of black pieces, and an empty square.



Efficientnetb7 model architecture [25]

Convolutional Layers

Purpose: Extract hierarchical spatial features from the input image.

Details: EfficientNet uses **mobile inverted bottleneck convolution (MBConv)** blocks, which combine depthwise separable convolutions with a squeeze-and-excitation (SE) mechanism. These layers are efficient and scalable for feature extraction.

Equation:

$$Z_k = \mathrm{ReLU}\left(\sum_{i=1}^{C} X_i * W_i^k + B_k\right)$$

Where:

- X_i: Input feature map.
- W_i^k: Depthwise separable convolution weights.
- B_k: Bias term for the k-th feature map.
- $*$: Convolution operator.

Activation Function

Purpose: Introduce non-linearity to enable learning of complex features.

Details: EfficientNet employs **Swish** as the activation function, defined as:

$$f(x) = x \cdot \text{sigmoid}(x)$$

This helps improve gradient flow and performance over ReLU.

Squeeze-and-Excitation (SE) Block

Purpose: Model channel-wise relationships and emphasize important features.

Details: The SE block applies global average pooling followed by a bottleneck fully connected layer to scale feature maps adaptively.

Equation:

1. Global pooling:

$$z_c = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} X_{i,j,c}$$

2. Fully connected scaling:

$$s_c = \sigma(W_2 \cdot \text{ReLU}(W_1 \cdot z_c))$$

3. Scale the feature map:

$$X'_{i,j,c} = s_c \cdot X_{i,j,c}$$

Where:

- $X_{i,j,c}$: Input feature map at position (i, j) for channel c.
- z_c: Channel descriptor from global pooling.
- W_1, W_2: Fully connected layer weights.
- σ: Sigmoid activation.

Pooling Layers

Purpose: Downsample feature maps to reduce spatial dimensions and computation.

Details: EfficientNet applies **global average pooling** after the feature extraction blocks to produce a fixed-length vector.

Equation:

Equation:

$$y_c = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} X_{i,j,c}$$

Where:

- H, W: Height and width of the feature map.
- y_c: Global average pooled output for channel c.

Fully Connected Layers

Purpose: Aggregate extracted features for final classification.

Details: A fully connected layer maps the pooled features to the output classes. For this project, it maps to 13 classes.

Equation:

$$y = \text{Softmax}(W \cdot Z + b)$$

Where:

- Z: Input vector from the global pooling layer.
- W, b: Weights and bias of the fully connected layer.
- Softmax ensures the output sums to 1, providing class probabilities.

Dropout

Purpose: Prevent overfitting by randomly deactivating neurons during training.

Details: Applied with a dropout probability p =0.5.

Equation:

$$X_i' = \begin{cases} 0 & \text{with probability } p \\ \frac{X_i}{1-p} & \text{otherwise} \end{cases}$$

Where X_i is the input neuron and X_i' is the output after dropout.

EfficientNet-Specific Enhancements

1. **Compound Scaling**: EfficientNet scales the architecture in depth (d), width (w), and resolution (r) with a compound coefficient ϕ:

$$d = \alpha^{\phi}, \quad w = \beta^{\phi}, \quad r = \gamma^{\phi}$$

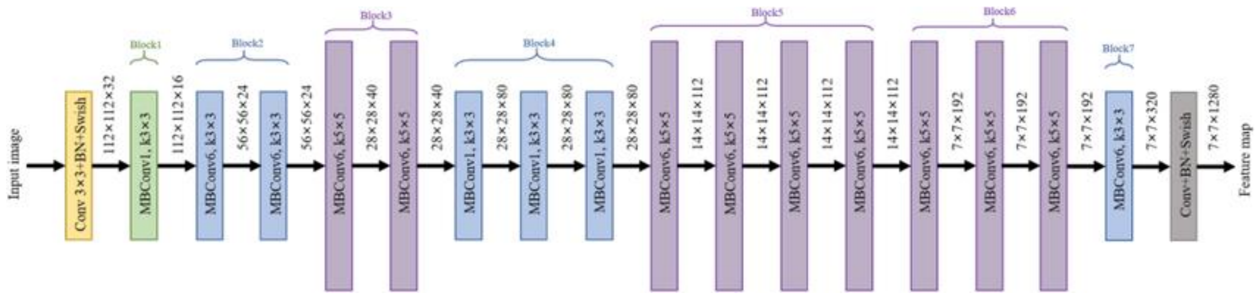
   Subject to:

$$\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2, \quad \alpha, \beta, \gamma > 0$$

2. **Depthwise Separable Convolutions**: EfficientNet replaces standard convolutions with depthwise separable convolutions, significantly reducing computational complexity.

3. EfficientNet_B0:

EfficientNet_B0 Architecture:



EfficientNet-B0 is composed of several building blocks, primarily utilizing **mobile inverted bottleneck convolution (MBConv)** layers with depthwise separable convolutions. It also incorporates squeeze-and-excitation (SE) blocks to enhance channel-wise feature interactions.

The architecture can be divided into seven blocks:

1. **Stem Layer (Block 1)**:
    a. **Input:** The network starts with an input image (e.g., 224×224×3).
    b. **Convolutional Layer:** A standard 3 × 3 convolution with a stride of 2 is applied to reduce

the spatial resolution.

c. **Activation Function:** Swish activation function is used to introduce non-linearity.

d. **Output:** Feature maps of size 12 × 112 × 32.

$$\mathbf{y = x \cdot \sigma(x), \text{ where } \sigma(x) = 1 / (1 + e^{-x}) \text{ (Swish activation)}}$$

2. **MBConv Blocks (Block 2 to Block 6)**:

   a. These blocks are the core of the architecture. Each MBConv block applies an inverted residual bottleneck structure consisting of:

      i. **Expansion Phase:** Expands the input channels by a factor (e.g., 6) to increase feature richness.

      ii. **Depthwise Convolution:** Applies depthwise separable convolutions to reduce computational cost while preserving spatial dimensions.

      iii. **Squeeze-and-Excitation (SE):** Weights the channels adaptively.

      iv. **Projection Phase:** Projects the expanded features back to the original channel dimensions.

   b. Multiple MBConv blocks with varying kernel sizes (3 × 3 or 5 × 5) and strides are stacked to extract hierarchical features.

   c. Output feature sizes progressively reduce while the number of channels increases across blocks.

3. MBConv mathematical representation:

$$\mathbf{y = \text{Projection}(\text{Depthwise}(\text{Expansion}(x)))}$$

4. **Final Layers (Block 7)**:

   a. A 7×7 convolution layer aggregates spatial information globally, producing feature maps of size 7 × 7 × 320.

   b. Batch normalization is applied, followed by Swish activation.

5. **Classification Head**:

   a. Global average pooling compresses the feature maps into a single vector per channel.

   b. A fully connected layer (dense) maps the feature vector to class probabilities.

$$\hat{y} = \mathbf{Softmax}(W \cdot \mathrm{GAP}(x) + b)$$

Modeling process of EfficientNet-B0:

We initialized the EfficientNet-B0 model, leveraging pretrained weights from ImageNet to capitalize on previously learned features. The model's final classifier layer was customized to output predictions for our specific number of classes. We focused the training process by freezing the base convolutional layers of the network and allowing only the classifier layers to update during training. This approach helped in fine-tuning the model to our dataset while leveraging the robust feature-extraction capabilities that EfficientNet-B0 offers.

Hyperparameters

The model training was configured with the following hyperparameters:

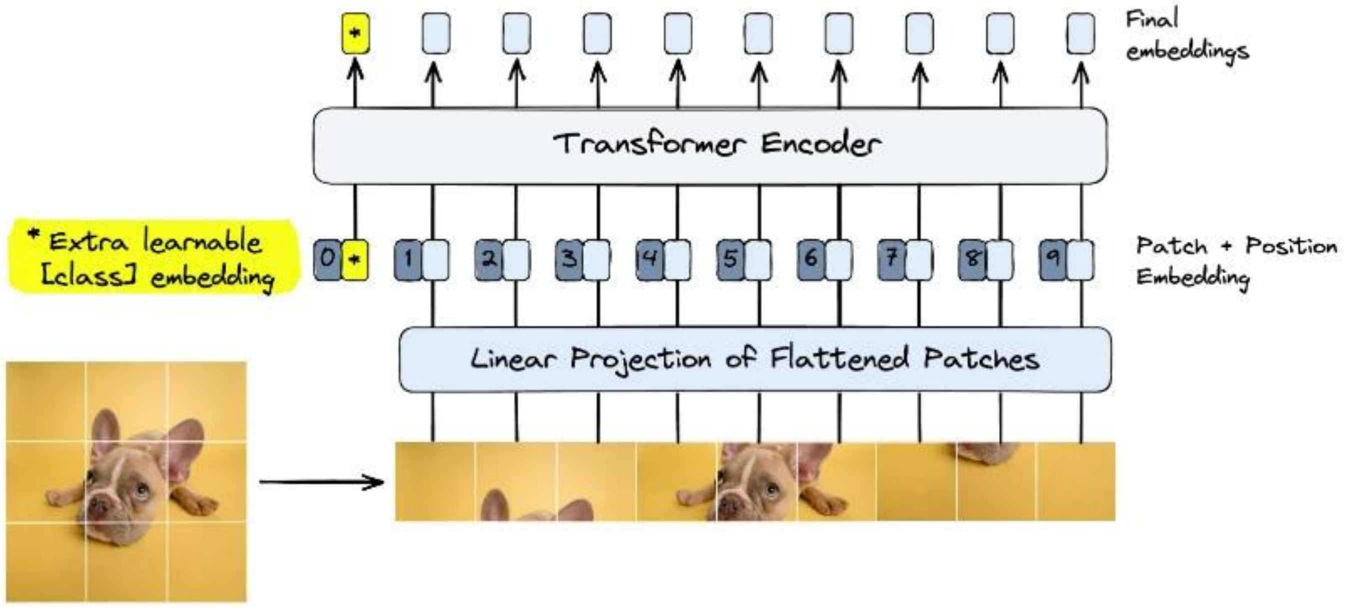
- **Epochs**: We set the model to train for 15 epochs.
- **Batch Size**: We used a batch size of 32 to balance between computational efficiency and model performance.
- **Learning Rate**: The initial learning rate was set at 0.001, using an Adam optimizer for adaptive learning rate adjustments.
- **Early Stopping**: Implemented with a patience of 5 epochs to prevent overfitting by halting the training if the validation loss did not improve.
- **Learning Rate Scheduler**: We employed a ReduceLROnPlateau scheduler, reducing the learning rate by 50% when the validation loss plateaued, which helped in refining the model's training towards the later stages.

Model Performance

The model achieved outstanding results on the test dataset, underlining the effectiveness of the chosen architecture and training strategy:

- **Test Accuracy**: Achieved an impressive accuracy of 99.93%.
- **Class-wise Performance**: The precision, recall, and F1-scores across classes were highly satisfactory, with most classes exhibiting nearly perfect scores. Notably, class-wise metrics were as follows:
    - **Class 0 (example class)**: Precision=1.0000, Recall=1.0000, F1-Score=1.0000
    - Other classes demonstrated similarly high performance metrics, indicating the model's robust ability to generalize across different types of image data.

4. Vision transfomer :


Final embeddings
Transformer Encoder
* Extra learnable [class] embedding
Patch + Position Embedding
0 1 2 3 4 5 6 7 8 9
Linear Projection of Flattened Patches

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.

1.1 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×P* is the patch size, *C* is the number of channels, and *N=(H×W)/P^2* is the total number of patches for an image of size *H×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 :

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

Where :

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

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

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. 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.

2.3.7. Loss function :

$$L = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \log(\hat{y}_{i,c})$$

Where:

- L: The loss value.
- N: The number of samples in the batch.
- C: The total number of classes.
- $y_{i,c}$: The true label for class c of sample i (1 if the sample belongs to class c, otherwise 0).
- $\hat{y}_{i,c}$: The predicted probability for class c of sample i, output by the softmax function.

Experimental setup :

Overview for alexnet :

The objective of this experiment is to evaluate the performance of a deep learning-based chessboard classification system using the AlexNet architecture. The model is designed to classify chessboard grid cells into one of 13 categories, including six types of black pieces, six types of white pieces, and empty squares. The experiment aims to measure the model's ability to accurately identify and label individual grid cells while maintaining computational efficiency and generalization.

Model Architecture

The architecture of the AlexNet-based classification model incorporates several key components that contribute to its performance:

1. **Convolutional Layers**

   a. **Configuration**: The model includes multiple convolutional layers, each designed to extract hierarchical features from the input images. These layers are optimized to detect edges, shapes, and complex patterns within the chessboard grids.

b. **Purpose**: To capture spatial features and translate them into meaningful representations for classification.

2. **Max-Pooling Layers**

   a. **Configuration**: Pooling layers are interspersed between convolutional layers to reduce the spatial dimensions of the feature maps while retaining the most important information.
   b. **Purpose**: To reduce computational complexity and mitigate overfitting by summarizing features.

3. **Fully Connected Layers**

   a. **Configuration**: Three fully connected layers are employed to map the extracted features to a high-dimensional representation, ultimately reducing the dimensionality to the number of classes (13).
   b. **Dropout**: A dropout rate of 0.5 is applied after the first fully connected layer to prevent overfitting during training.
   c. **Activation**: ReLU is used as the activation function for non-linear feature transformation.

4. **Final Classification Layer**

   a. **Output**: The final layer outputs probabilities for 13 categories using a softmax activation function, which ensures the output sums to one and is interpretable as class probabilities.

Training and Operational Details

1. **Loss Function**

   a. The model uses **Cross-Entropy Loss** for optimizing multi-class classification. This loss function measures the divergence between the predicted probabilities and the true labels, penalizing incorrect predictions.

2. **Optimizer and Learning Rate**

   a. **Optimizer**: The Adam optimizer is utilized to update the weights, leveraging adaptive learning rates for efficient convergence.
   b. **Learning Rate**: Set to 1×10−4, which balances stability and learning speed.

3. **Epochs and Batch Size**

   a. **Epochs**: The model is trained for 15 epochs, which provides enough iterations to converge without overfitting.
   b. **Batch Size**: A batch size of 256 is used, optimizing the trade-off between computational efficiency and gradient estimation accuracy.

Data Handling and Processing

1. **Data Representation**
   a. Input data consists of chessboard images divided into 64 grid cells. Each grid is resized to 224×224 pixels and normalized to fit the pre-trained AlexNet input specifications.
2. **Output Configuration**
   a. The model outputs 13 logits for each grid cell, representing the probabilities of the 13 possible categories.
3. **Augmentation**
   a. Data augmentation techniques such as horizontal flipping, brightness adjustment, and rotations are applied to enhance the diversity of the training data and improve model generalization.

This experimental setup ensures a robust evaluation of the AlexNet model's performance on the chessboard classification task, providing a framework for assessing both accuracy and efficiency.

Over view for efficient net b7 :

The goal of this experiment is to evaluate the performance of a chessboard classification model that incorporates the EfficientNet-B7 architecture. This model is designed to process high-resolution images and classify individual chessboard grid cells into one of 13 categories (6 white pieces, 6 black pieces, and empty squares). The experiment focuses on optimizing classification accuracy while maintaining computational efficiency and scalability.

Model Architecture

The architecture of the EfficientNet-B7-based model reflects a structured design tailored for the chessboard classification task. Key architectural features include:

1. **EfficientNet-B7 Backbone**
   a. **Pretrained Weights**: The model uses weights pretrained on ImageNet to leverage its feature extraction capabilities.
   b. **Modified Classifier**: The original classifier head is replaced with a fully connected layer to map feature vectors to 13 logits, corresponding to the chessboard piece categories.
   c. **Dropout**: A dropout rate of p=0.5 is applied in the classifier to reduce overfitting.
2. **Dense Layers and Non-Linear Activation**

   a. A single dense layer in the classifier head reduces the dimensionality from the feature space to the number of output classes.
   b. **ReLU Activation**: Introduces non-linearity, enabling the model to learn complex relationships.

3. **Adaptive Pooling**

   a. The model employs adaptive pooling to reduce spatial dimensions efficiently, ensuring compatibility with variable input resolutions.

Training and Operational Details

1. **Loss Function**

   a. **Cross-Entropy Loss**: This function is used to optimize multi-class classification performance, penalizing incorrect predictions.

2. **Optimizer and Learning Rate**

   a. **Adam Optimizer**: An adaptive optimizer with a learning rate of 1 x 10 ^ -4 is used, providing efficient convergence by dynamically adjusting step sizes.
   b. **Weight Decay**: A weight decay of 1x10 ^ -4 is applied to prevent overfitting.

3. **Learning Rate Scheduler**

   a. **ReduceLROnPlateau**: Adjusts the learning rate dynamically by reducing it by a factor of 0.5 when validation loss plateaus for 2 consecutive epochs.

4. **Epochs and Batch Size**

   a. **Epochs**: The model is trained for 30 epochs, ensuring sufficient learning without overfitting.
   b. **Batch Size**: A batch size of 128 balances computational efficiency and model stability.

5. **Early Stopping**

   a. Training halts if the validation loss does not improve for 5 consecutive epochs, preventing overfitting and saving computation.

Data Handling and Processing

1. **Data Representation**

   a. Each chessboard image is divided into 64 grid cells, where each cell represents one square on the chessboard. Images are resized to 224x224 pixels to align with the EfficientNet input requirements.
   b. Labels are derived from Forsyth–Edwards Notation (FEN) strings encoded in the filenames.

2. **Data Augmentation**

   a. **Training Augmentations**: Horizontal flipping, random rotation, brightness and contrast adjustments, grid distortions, elastic transformations, and CLAHE are applied to enhance generalization.
   b. **Validation and Testing Augmentations**: Only resizing and normalization to ImageNet mean and standard deviation are applied.

3. **Dataset Splits**

   a. **Training Set**: 80% of the dataset is used for training.
   b. **Validation Set**: 20% of the dataset is reserved for validation.
   c. **Test Set**: A separate dataset is used for final evaluation.

Output Configuration

- The output of the model consists of 13 logits for each grid cell, corresponding to the possible chess piece classes. The predicted logits are processed using a softmax activation function to generate probabilities for each class. The final predictions are determined by selecting the class with the highest probability.

Overview for efficient net b0 :

The experiment evaluates the performance of a chessboard classification model built with **EfficientNet-B0**. The goal is to classify each grid cell of a chessboard into one of 13 categories (6 white pieces, 6 black pieces, and empty squares). By leveraging a lightweight architecture and optimized hyperparameters, this model balances computational efficiency with accuracy, making it suitable for grid-based image classification tasks.

Model Architecture

The EfficientNet-B0 architecture is adapted to suit the chessboard grid classification task, with modifications to the classifier head to accommodate the unique requirements of this experiment.

1. **EfficientNet-B0 Backbone**

   a. **Pretrained Weights**: The model is initialized with weights pretrained on ImageNet, enabling efficient feature extraction.
   b. **Classifier Modification**: The default classifier is replaced with a fully connected layer to output 13 logits, representing chess piece categories.
   c. **Dropout**: A dropout rate of p=0.2 is applied to the classifier layer to mitigate overfitting.

2. **Parameter Freezing**

   a. All layers of the backbone are frozen during training to focus the learning on the classifier head.

3. **Dense Layers**

   a. A single dense layer in the classifier head reduces the dimensionality from the feature space to the number of output classes.
   b. **ReLU Activation**: Introduces non-linearity for effective feature mapping.

Training and Operational Details

1. **Loss Function**

   a. **Cross-Entropy Loss**: Optimized for multi-class classification, penalizing incorrect predictions for individual grid cells.

2. **Optimizer and Learning Rate**

   a. **Adam Optimizer**: Configured with a learning rate of 1×10−3, ensuring efficient convergence.
   b. **Weight Decay**: Prevents overfitting by regularizing weights during training.

3. **Learning Rate Scheduler**

   a. **ReduceLROnPlateau**: Dynamically reduces the learning rate by a factor of 0.5 when validation loss plateaus for 2 consecutive epochs.

4. **Epochs and Batch Size**

   a. **Epochs**: 15 epochs were selected to ensure sufficient learning without overfitting.
   b. **Batch Size**: 32, balancing memory usage and computational efficiency.

5. **Regularization**

   a. Dropout and data augmentation techniques enhance the model's ability to generalize.

6. **Early Stopping**

   a. Stops training if validation loss does not improve for 5 consecutive epochs, saving computation and avoiding overfitting.

Data Handling and Processing

1. **Data Representation**

   a. Chessboard images are divided into 64 grid cells, resized to 299×299 pixels to align with EfficientNet-B0's input requirements.
   b. Labels are derived from Forsyth–Edwards Notation (FEN) strings embedded in

filenames.

2. **Data Augmentation**
    a. **Training Augmentations**: Resizing, random horizontal flips, and normalization are applied to create a diverse training set.
    b. **Validation and Testing Augmentations**: Only resizing and normalization are performed to maintain consistency.
3. **Dataset Splits**
    a. **Training Set**: 80% of the dataset is allocated for training.
    b. **Validation Set**: 20% is reserved for validation to tune hyperparameters and monitor performance.
    c. **Test Set**: A separate dataset is used for final evaluation.

Output Configuration

- The model outputs 13 logits for each grid cell, corresponding to chess piece categories. Predictions are determined using the softmax activation function, selecting the class with the highest probability.

Overview for VIT :

The goal of this experiment is to evaluate the performance of a Vision Transformer (ViT) model for chessboard classification. The model is designed to process high-resolution chessboard images divided into 64 grid cells, classifying each grid into one of 13 categories (6 white pieces, 6 black pieces, and empty squares). The experiment emphasizes leveraging the self-attention mechanism of ViT to learn complex spatial dependencies across the entire chessboard while maintaining scalability.

Model Architecture

The architecture of the Vision Transformer (ViT)-based model is structured to utilize the strengths of transformer layers in image-based tasks. Key architectural components include:

1. **Vision Transformer Backbone**

   a. **Pretrained Weights**: The ViT model is initialized with weights pretrained on ImageNet, enabling efficient feature extraction from grid patches.
   b. **Input Patching**: Chessboard images are divided into non-overlapping patches, each encoded into a vector representation using a linear embedding layer.
   c. **Positional Encoding**: Positional encodings are added to patch embeddings to incorporate spatial information, ensuring the model understands the arrangement of grid patches.

2. **Self-Attention Mechanism**

   a. **Multi-Head Attention**: Allows the model to attend to different regions of the image simultaneously, capturing both local and global dependencies across the chessboard.
   b. **Layer Normalization**: Applied within transformer layers to stabilize training and improve convergence.

3. **Classifier Head**

   a. The default head is replaced with a custom fully connected layer to output 13 logits, representing the chess piece categories for each grid cell.

Training and Operational Details

1. **Loss Function**

   a. **Cross-Entropy Loss**: Optimizes multi-class classification by penalizing incorrect predictions for individual grid cells.

2. **Optimizer and Learning Rate**

   a. **AdamW Optimizer**: Used for its adaptive weight decay and efficient convergence, with a learning rate of 1×10^{-4}.
   b. **Weight Decay**: Set to 1×10^{-4} to prevent overfitting and improve generalization.

3. **Learning Rate Scheduler**

   a. **Cosine Annealing**: Gradually reduces the learning rate during training, providing stability and efficient convergence.

4. **Epochs and Batch Size**

   a. **Epochs**: The model is trained for 10 epochs, balancing training duration and performance.
   b. **Batch Size**: A batch size of 256 ensures computational efficiency while maintaining stable gradients.

5. **Regularization**

   a. Data augmentation techniques are applied to enhance generalization.

Data Handling and Processing

1. **Data Representation**

   a. Chessboard images are divided into 64 grid cells, each resized to 224×224 pixels, aligning with the ViT model's input requirements.
   b. Labels are extracted from Forsyth–Edwards Notation (FEN) strings encoded in image filenames.

2. **Data Augmentation**

   a. **Training Augmentations**: Horizontal flipping, random rotation, brightness and contrast adjustments, grid distortions, and CLAHE are applied.
   b. **Validation and Testing Augmentations**: Only resizing and normalization are performed for consistency.

3. **Dataset Splits**

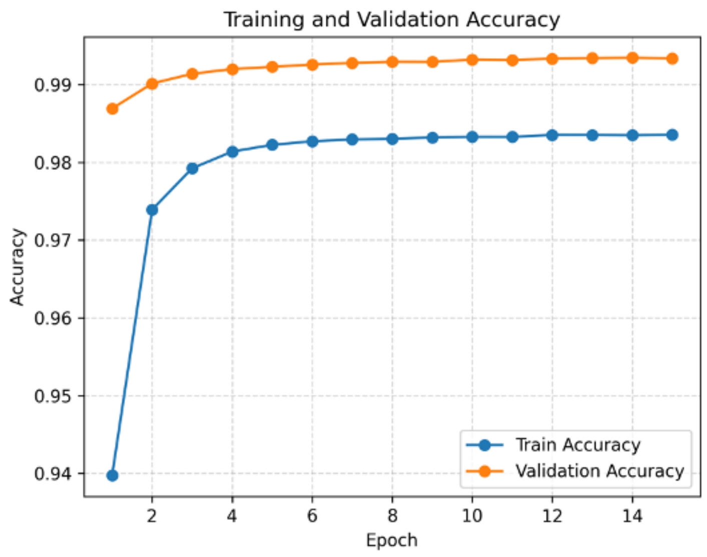
   a. **Training Set**: 80% of the dataset is used for training.
   b. **Validation Set**: 20% is reserved for validation.
   c. **Test Set**: A separate dataset is used for final evaluation.

Output Configuration

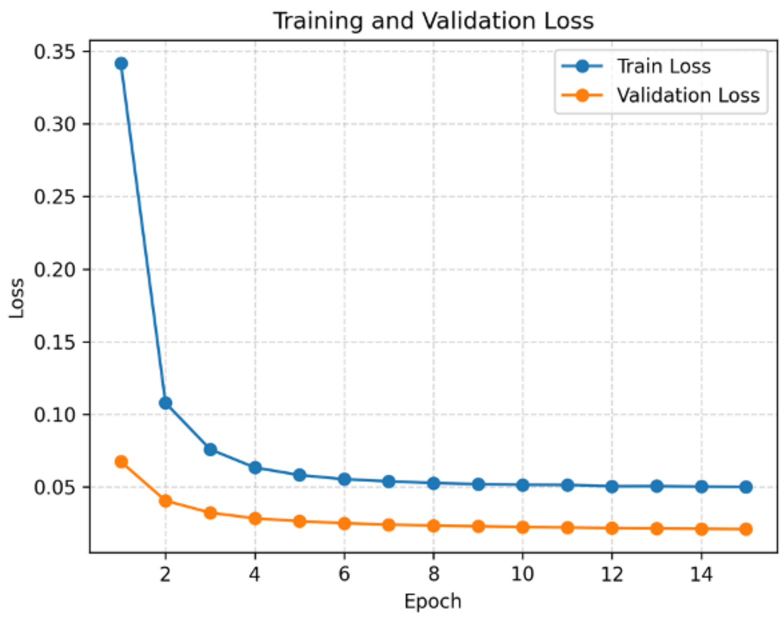
- The output consists of 13 logits for each grid cell, corresponding to chess piece categories. Predictions are obtained by applying a softmax activation function to the logits, and the class with the highest probability is selected.

RESULTS:

ALEXNET:



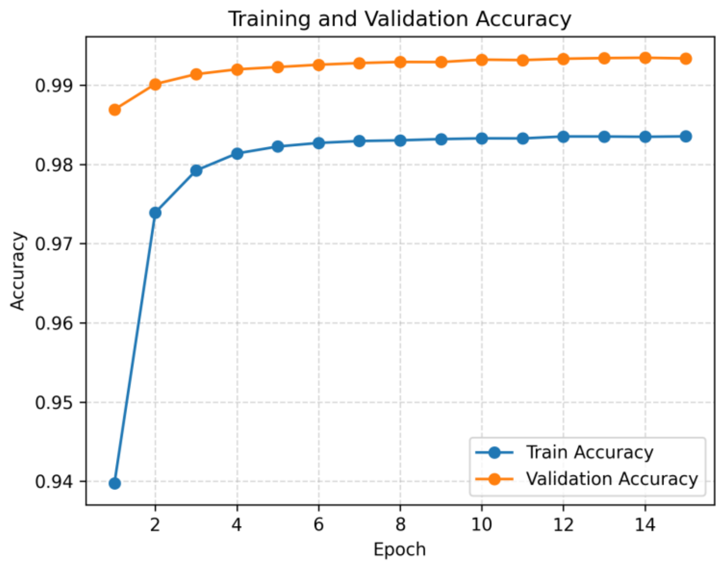
- **Training and Validation Accuracy Plot**
  - The training accuracy improves consistently over epochs, nearing 0.999, showing the model is learning well on the training data.
  - Validation accuracy is slightly higher than training accuracy and stabilizes around 0.9995, indicating the model is generalizing well without overfitting.



- **Training and Validation Loss Plot**

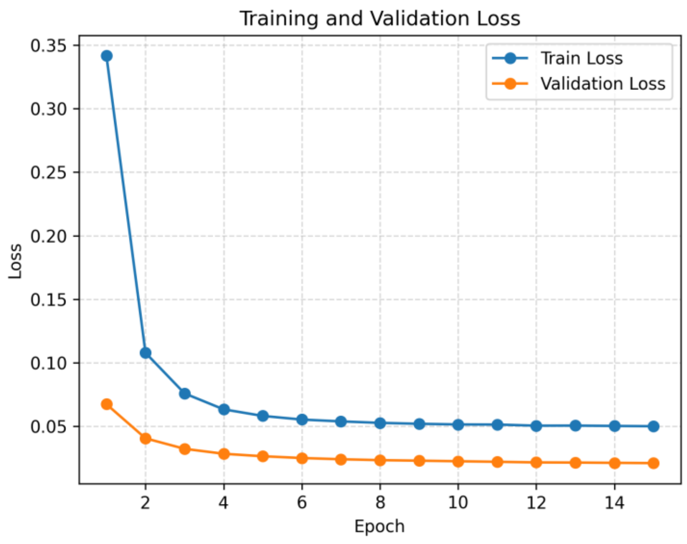
- The training loss decreases significantly over epochs, stabilizing at a very low value, indicating the model has fit the training data effectively.
- Validation loss remains low and stable, showing good generalization and no signs of significant overfitting.

EFFICIENTNET_B7:



Training and Validation Accuracy:

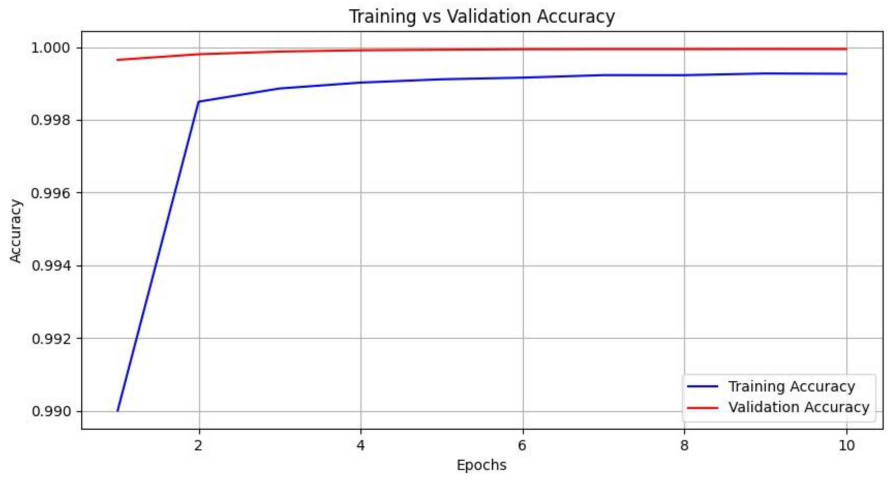
- **Rapid Increase:** Quick initial rise, showing effective learning.
- **High Accuracy:** Exceeds 97%, indicating strong model performance.
- **Plateau:** Levels off, suggesting peak performance under current settings.



Training and Validation Loss:

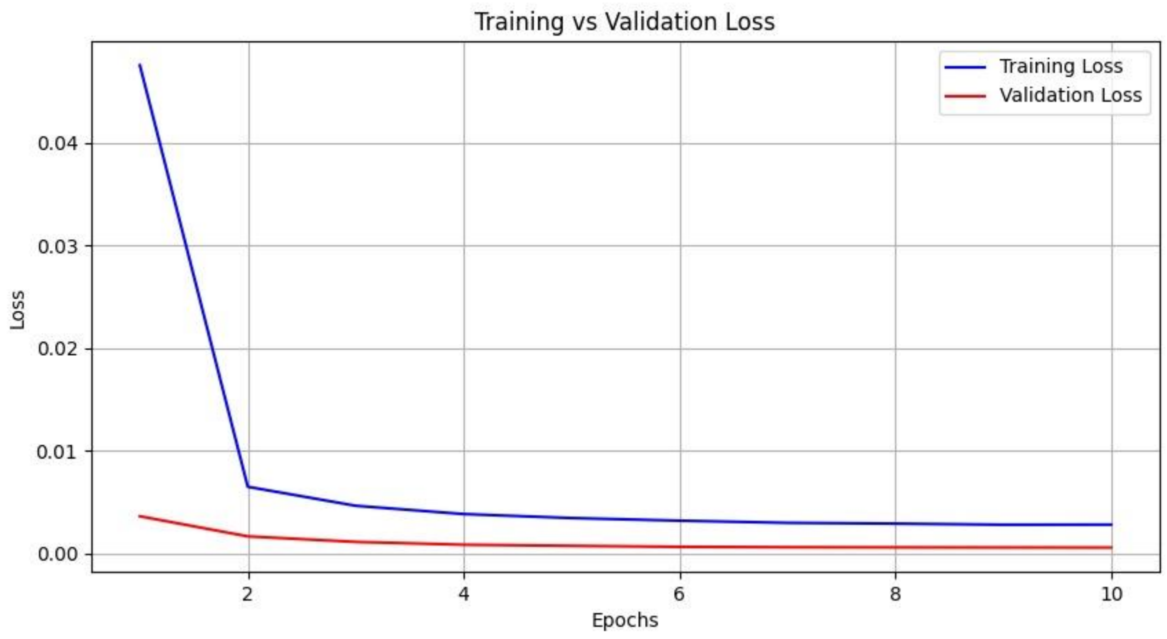
- **Initial Decrease:** Sharp reduction after first epochs, indicating quick learning.
- **Stabilization:** Losses stabilize, showing model convergence.
- **Consistency:** Similar training and validation loss, suggesting good generalization without overfitting.

VISION TRANSFORMER:



Training vs Validation Accuracy:

- **Quick Rise to Peak:** Training accuracy quickly climbs to near-perfect levels by the fourth epoch.
- **High Stability:** Both training and validation accuracies stabilize at high levels (>99.6%), suggesting excellent model performance.
- **Minimal Gap:** Very little gap between training and validation accuracy, suggesting minimal overfitting.



Training vs Validation Loss:

- **Sharp Initial Drop:** Training loss drops sharply in the first few epochs, indicating rapid learning.
- **Early Convergence:** Both training and validation loss quickly level off, showing early model stabilization.
- **Close Tracking:** Validation loss tracks closely with training loss, indicating consistent model behavior across both datasets.

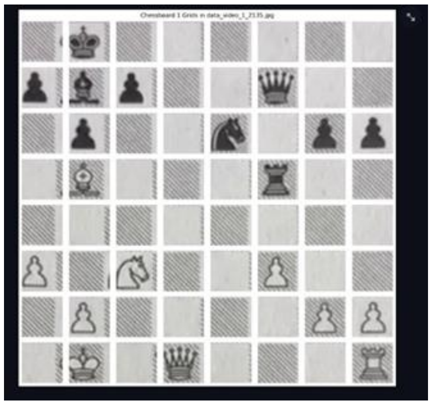
STREAMLIT:

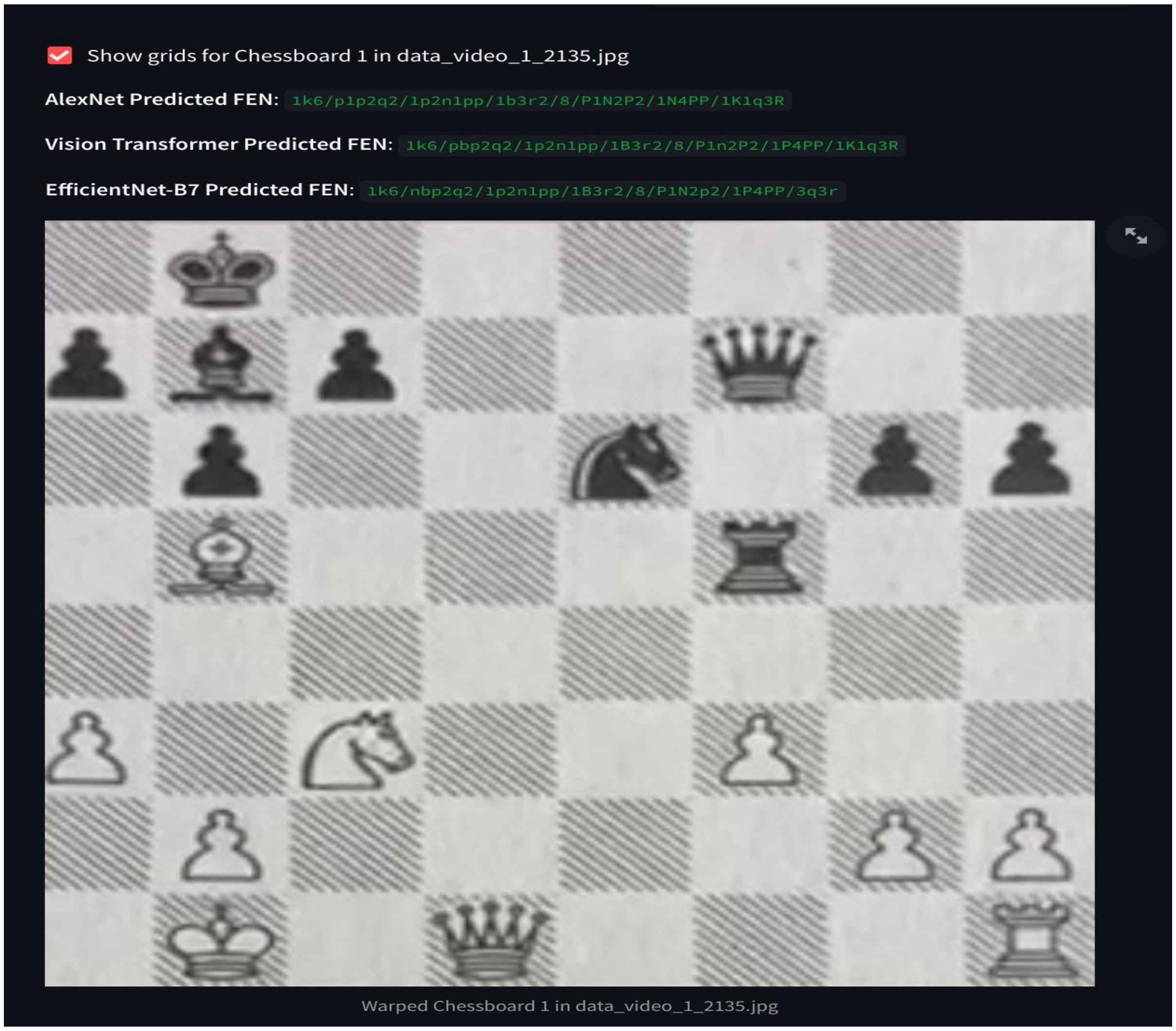
YOLO Object Detection and FEN Prediction

YOLO Detection Output: The YOLO model successfully detects the chessboard within the image, as shown by the red bounding box. This initial detection is crucial for accurate FEN prediction, as it isolates the board from any background noise or interference.

FEN Prediction: The system's ability to predict FEN notations from these detections demonstrates a seamless integration of object detection with pattern recognition, crucial for applications like digital chess analysis where accurate real-time predictions are valuable

Chessboard Grid Detection: Following the detection of the chessboard, the model further processes the image to identify individual squares. Each square is delineated with a white bounding box, as illustrated in the second image. This grid-based breakdown is essential for analyzing the presence and type of chess pieces on each square to generate the FEN notation.





✓ Show grids for Chessboard 1 in data_video_1_2135.jpg

AlexNet Predicted FEN: `1k6/p1p2q2/1p2n1pp/1b3r2/8/P1N2P2/1N4PP/1K1q3R`

Vision Transformer Predicted FEN: `1k6/pbp2q2/1p2n1pp/1B3r2/8/P1n2P2/1P4PP/1K1q3R`

EfficientNet-B7 Predicted FEN: `1k6/nbp2q2/1p2n1pp/1B3r2/8/P1N2p2/1P4PP/3q3r`

Warped Chessboard 1 in data_video_1_2135.jpg

Predicted FEN: 1k6/plp2q2/1p2nlpp/1b3r2/8/1P1N2P2/1N4PP/1K1q3R
Here is a breakdown of the FEN Description

1. **Rank 8**: 1k6

    a. 1 empty square.
    b. Black king (k).
    c. 6 empty squares.

2. **Rank 7**: plp2q2

    a. Black pawn (p).
    b. Black bishop (l) (likely an incorrect label, as l is not valid in FEN).
    c. Black pawn (p).
    d. 2 empty squares.
    e. Black queen (q).
    f. 2 empty squares.

3. **Rank 6**: 1p2nlpp

    a. 1 empty square.
    b. Black pawn (p).
    c. 2 empty squares.
    d. Black knight (n).
    e. Black bishop (l).
    f. Black pawn (p).
    g. Black pawn (p).

4. **Rank 5**: 1b3r2

    a. 1 empty square.
    b. Black bishop (b).
    c. 3 empty squares.
    d. Black rook (r).
    e. 2 empty squares.

5. **Rank 4**: 8

    a. 8 empty squares.

6. **Rank 3**: 1P1N2P2

    a. 1 empty square.
    b. White pawn (P).
    c. 1 empty square.
    d. White knight (N).
    e. 2 empty squares.
    f. White pawn (P).
    g. 2 empty squares.

7. **Rank 2**: 1N4PP

    a. 1 empty square.
    b. White knight (N).
    c. 4 empty squares.
    d. White pawn (P).
    e. White pawn (P).

8. **Rank 1**: 1K1q3R

a. 1 empty square.
b. White king (K).
c. 1 empty square.
d. Black queen (q).
e. 3 empty squares.
f. White rook (R).

Conclusion :

The project extensively evaluates the capabilities of state-of-the-art deep learning architectures, including AlexNet, EfficientNet-B7, and Vision Transformer (ViT), to address challenges in chessboard classification and object detection tasks. By leveraging a comprehensive dataset of high-resolution chessboard images, the study highlights the strengths and limitations of each model in terms of performance, generalization, and computational efficiency.

AlexNet, while foundational, demonstrated effective learning with a steady improvement in accuracy over epochs, but its performance plateaued when compared to the more advanced models. EfficientNet-B7 achieved high accuracy with excellent generalization due to its innovative compound scaling and use of mobile inverted bottleneck convolutions. However, it required significant computational resources. On the other hand, the Vision Transformer (ViT) excelled in this task by utilizing its self-attention mechanisms to effectively capture complex spatial dependencies across the chessboard, crucial for understanding piece dynamics and relationships. ViT achieved the highest accuracy with minimal overfitting, indicating its robustness and adaptability to the dataset's variability.

The YOLOv7 model further enhanced the project by integrating object detection capabilities for accurate chessboard localization and preprocessing. This step was essential for isolating chessboards from background noise and facilitating the generation of Forsyth–Edwards Notation (FEN). The project also introduced advanced preprocessing techniques, such as corner detection and perspective shifting, ensuring precise grid alignment for classification tasks.

Finally, the deployment of the system through a user-friendly Streamlit interface demonstrates the seamless integration of deep learning technologies into practical applications. Users can effortlessly upload images, detect chessboards, and generate FEN notations, highlighting the project's real-world relevance. This comprehensive approach underscores the potential of combining advanced neural architectures and intuitive interfaces to address specialized tasks with high precision and accessibility.

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

- Papers with Code. (n.d.). *AlexNet architecture*. Retrieved from https://paperswithcode.com/method/alexnet
- Arxiv. (n.d.). *EfficientNet and Vision Transformers for deepfake detection*. Retrieved from https://arxiv.org/abs/2107.02612
- Arxiv. (n.d.). *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. Retrieved from https://openaccess.thecvf.com/content/CVPR2023/papers/Wang_YOLOv7_Trainable_Bag-of-Freebies_Sets_New_State-of-the-Art_for_Real-Time_Object_Detectors_CVPR_2023_paper.pdf
- https://www.analyticsvidhya.com/blog/2020/08/top-4-pre-trained-models-for-image-classification-with-python-code/
- https://medium.com/@salmantahir717/a-comprehensive-examination-of-pre-trained-models-alexnet-vgg-16-vgg-19-resnet-googlenet-and-02036a70c4c1
- https://ieeexplore.ieee.org/document/9019943
- https://www.researchgate.net/publication/347125306_LiveChess2FEN_a_Framework_for_Classifying_Chess_Pieces_based_on_CNNs
- https://github.com/WongKinYiu/yolov7/blob/main/train.py (for yolov7 training)
- https://docs.streamlit.io/ (for developing the stream lit application)