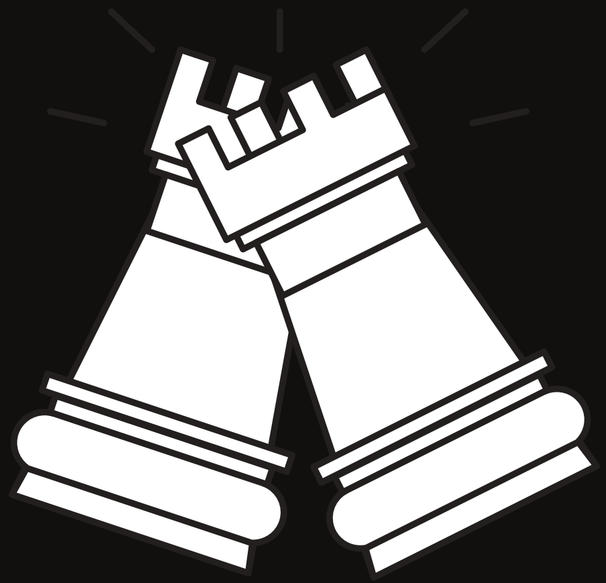
DEEP LEARNING



# GENERATING FEN-NOTATIONS FROM CHESS BOARDS IMAGES

GROUP 4
Sudhanshu Dotel
Bhanu Sai Praneeth Sarva
Ashwin Muthuraman
Bhoomika Nanjaraja

# INTRODUCTION

**Project Overview**

Identifying chessboard positions using deep learning

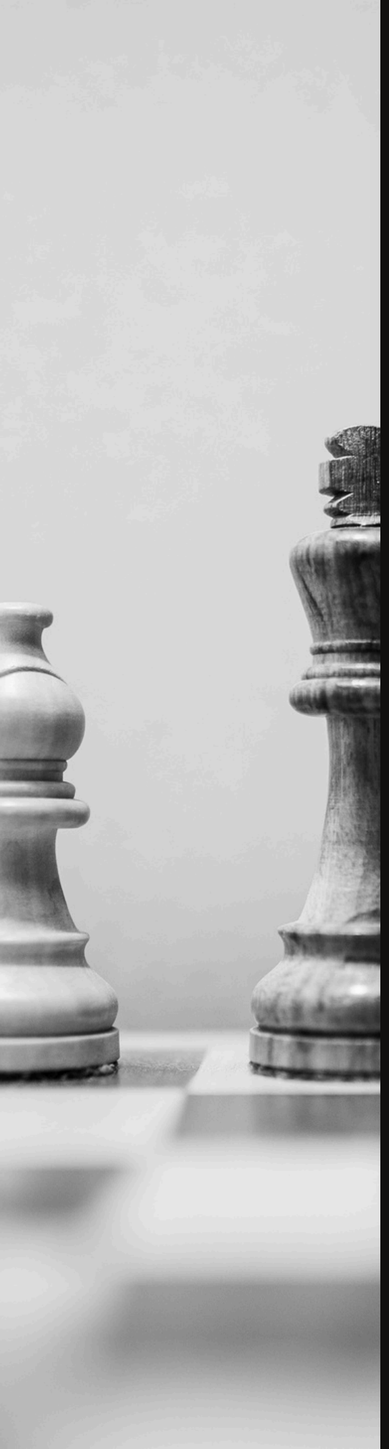
**Objective**

To train a model that accurately identifies and classifies chessboard positions into labeled grids using FEN notation

**Significance**

FEN notations are used in:

- Chess Engines and Software: Stockfish, Leela Chess Zero
- Online Chess Platforms: Chess.com, Lichess
- Game Databases



# What is a FEN Notation?

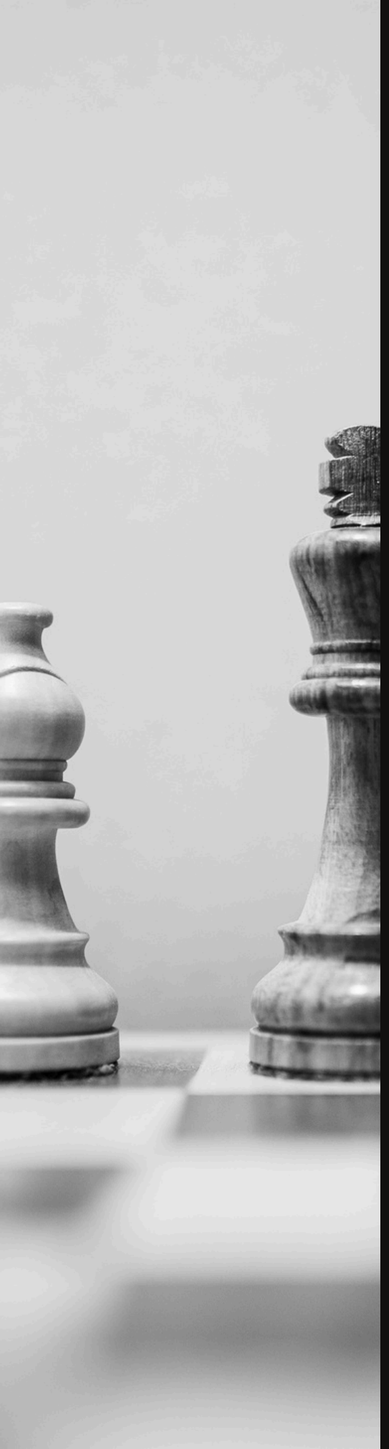

- **Black Pieces (Lowercase)**

- r: Black Rook
- n: Black Knight
- b: Black Bishop
- q: Black Queen
- k: Black King
- p: Black Pawn

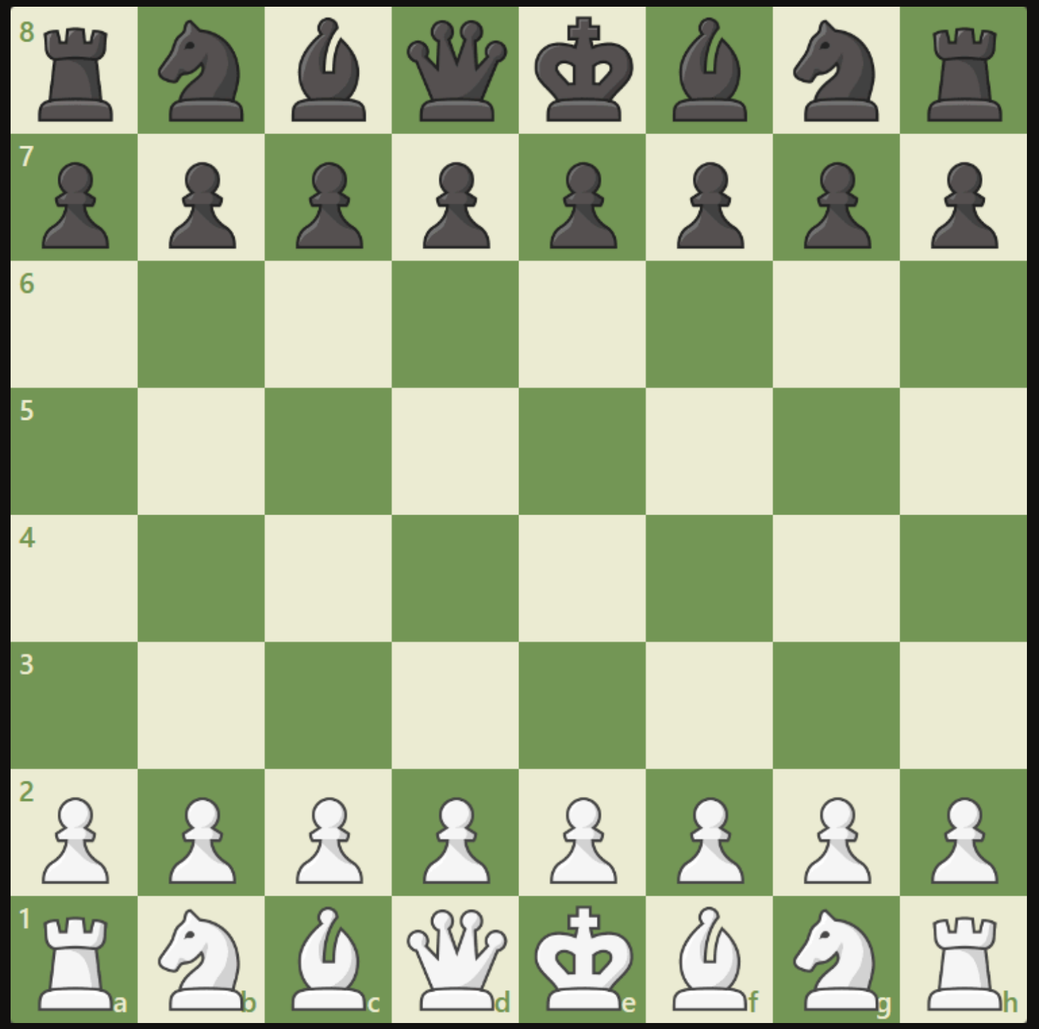
- **WHITE PIECES (UPPERCASE)**

- R: WHITE ROOK
- N: WHITE KNIGHT
- B: WHITE BISHOP
- Q: WHITE QUEEN
- K: WHITE KING
- P: WHITE PAWN

**FEN Notation: r1bk3r-p2pBpNp-n4n2-1p1NP2P-6P1-3P4-P1P1K3-q5b1**



# What is a FEN Notation?




- **Black Pieces (Lowercase)**

- **r: Black Rook**
- **n: Black Knight**
- **b: Black Bishop**
- **q: Black Queen**
- **k: Black King**
- **p: Black Pawn**

- **WHITE PIECES (UPPERCASE)**

- **R: WHITE ROOK**
- **N: WHITE KNIGHT**
- **B: WHITE BISHOP**
- **Q: WHITE QUEEN**
- **K: WHITE KING**
- **P: WHITE PAWN**

**FEN Notation: rnbqkbnr-pppppppp-8-8-8-8-PPPPPPPP-RNBQKBNR**

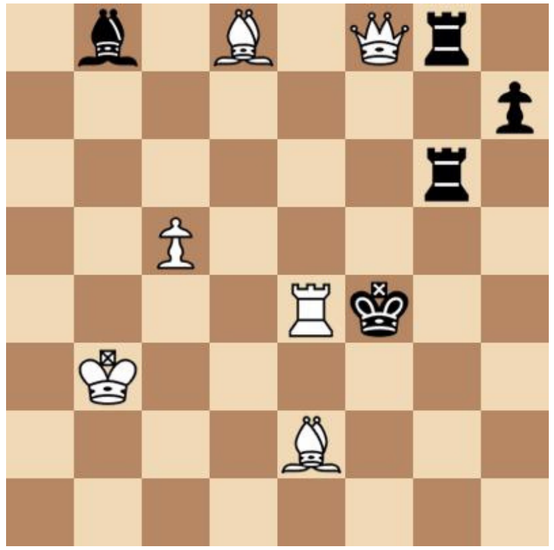
# ABOUT THE DATASET

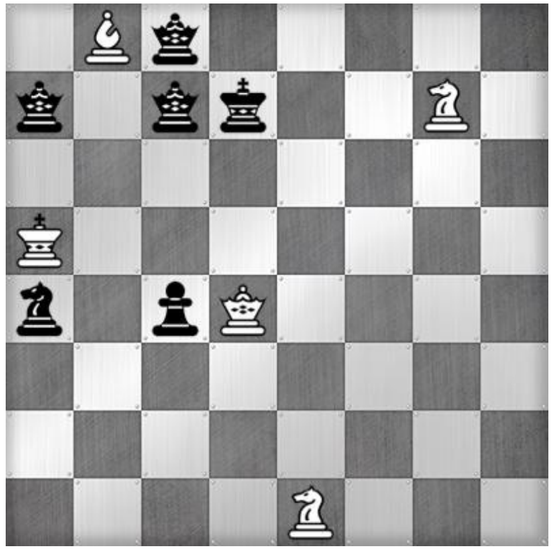
- **Dataset from Kaggle**
  **https://www.kaggle.com/datasets/koryakinp/chess-positions/data**

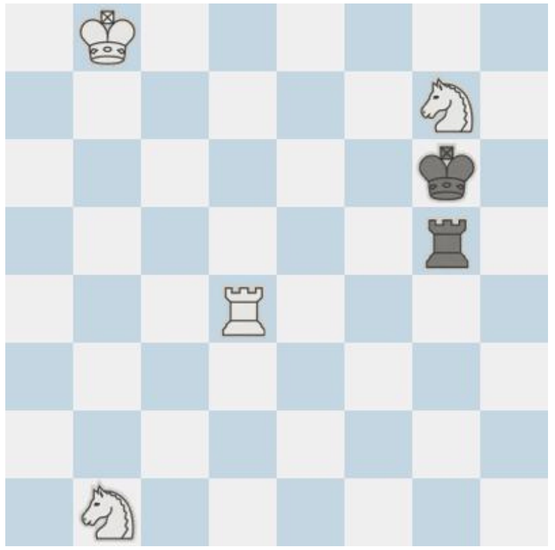
- **100k images ----> 80k training, 20k testing**
- **All images are 400 x 400 pixels**

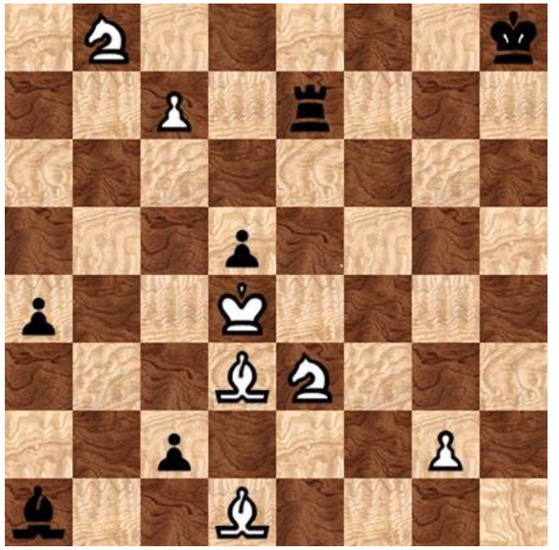
- **5-15 pieces (2 kings and 3-13 pawns/pieces)**
- **28 styles of chess boards**
- **32 styles of chess pieces**
- **totaling 896 board/piece style combinations.**
- **probability distribution:**
  - **30% for Pawn**
  - **20% for Bishop**
  - **20% for Knight**
  - **20% for Rook**
  - **10% for Queen**
  - **2 Kings are guaranteed to be on the board.**

# SAMPLE DATASET









# DATA PRE-PROCESSING

- Splitting images into grids.
- Image Augmentation
- Normalization and resizing
- FEN to label conversion.

# DATA PRE-PROCESSING

- **Splitting images into grids.**
  - **Divide each chessboard image into 64 smaller grids (8x8 board structure)**
  - **Extract grid segments using slicing and store them for labeling and training.**
- **So each grid becomes a single image classification task**
- **Therefore total classification images become:**
  - **Training: 80,000 x 64 = 5,120,000**
  - **Testing:  20,000 x 64 = 1,280,000**

# DATA PRE-PROCESSING

## FEN to label conversion:

- **FEN Overview:**
  - **Encodes chessboard state using a compact string format.**
  - **Example: "rnbqkbnr-pppppppp-8-8-8-8-PPPPPPPP-RNBQKBNR".**
- **Conversion Process:**
  - **Split FEN string into rows using -**
  - **Map characters (p, P, k, etc.) to numeric labels (e.g., 1 for black pawn, 2 for white pawn, 0 for empty grid).**
  - **(Total- 13 labels)**
  - **Replace numbers (empty squares) with consecutive 0s.**
- **Output:**
  - **Flattened list of 64 labels, one for each square.**
  - **Example: [5, 7, 3, 9, ..., 0, 12].**

# DATA PRE-PROCESSING

## Augmentations:

- Horizontal Flip (p=0.5)
- Rotation (limit=15°, p=0.5)
- Random Brightness & Contrast (p=0.5)
- Shift, Scale, Rotate (p=0.5)
- Grid Distortion (p=0.3)
- Elastic Transform (p=0.3)
- CLAHE (p=0.5)

## Significance:

- Improve model robustness by simulating real-world variations in chessboard images.

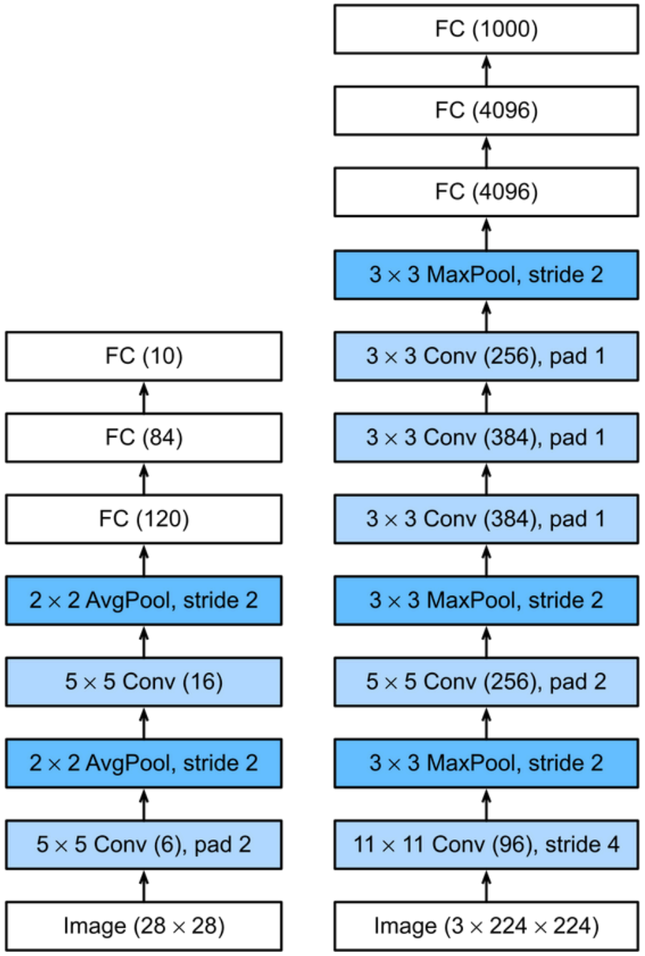
# DATA PRE-PROCESSING

## Resizing and Normalizing Images

- **Normalization:**
  - **Pixel values normalized using ImageNet statistics:**
  - **Mean: [0.485, 0.456, 0.406]**
  - **Std: [0.229, 0.224, 0.225].**
  - **Ensures compatibility with pre-trained model weights.**
- **Resizing:**
  - **Each grid resized to compatible size based on the input requirements of each model**

# MODEL ARCHITECTURE ALEXNET

## AlexNet:




```python
def get_model(pretrained: bool = True, num_classes: int = cfg.OUTPUTS_A) -
    model = models.alexnet(pretrained=pretrained)

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

    model.avgpool = nn.AdaptiveAvgPool2d((6, 6))
    model.classifier = nn.Sequential(
        nn.Dropout(),
        nn.Linear(256 * 6 * 6, 4096),
        nn.ReLU(inplace=True),
        nn.Dropout(),
        nn.Linear(4096, 4096),
        nn.ReLU(inplace=True),
        nn.Linear(4096, num_classes),
    )

    return model.to(cfg.DEVICE)
```

# MODEL DEFINITION ALEXNET

**Loss Function (CrossEntropyLoss): Measures how well the model's predictions match the true class labels.**

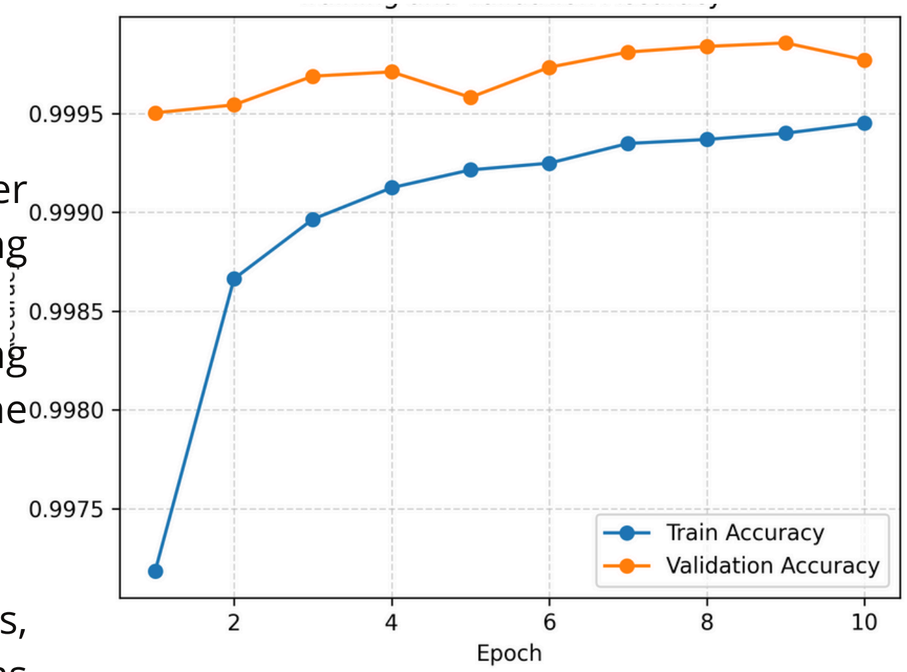
**Optimizer (Adam): Optimizes the model parameters using the Adam algorithm, which adapts learning rates for each parameter based on the gradients.**

**Learning Rate Scheduler (ReduceLROnPlateau): Adjusts the learning rate dynamically based on the validation loss. If the validation loss stops improving for 2 consecutive epochs, the learning rate is reduced by a factor of 0.5, potentially helping the model converge better.**

# EVALUATION PLOTS

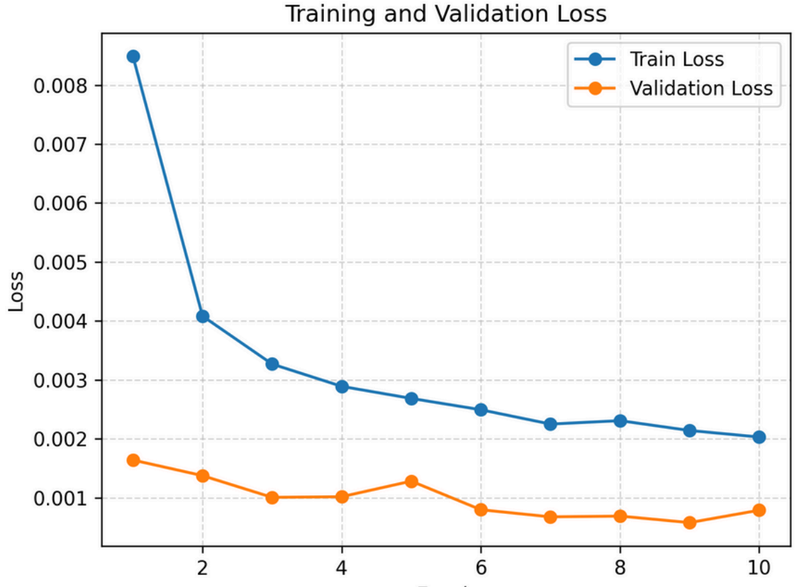
**1. Training and Validation Accuracy Plot**

- 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.

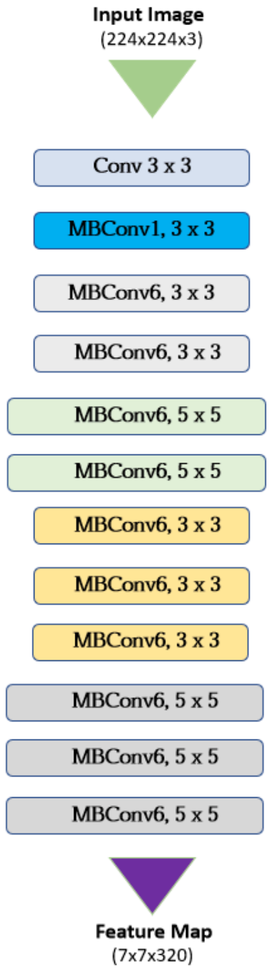



**2. Training and Validation Loss Plot**

- 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.



# MODEL ARCHITECTURE
# EFFICIENT NET B7




EfficientNet Architecture

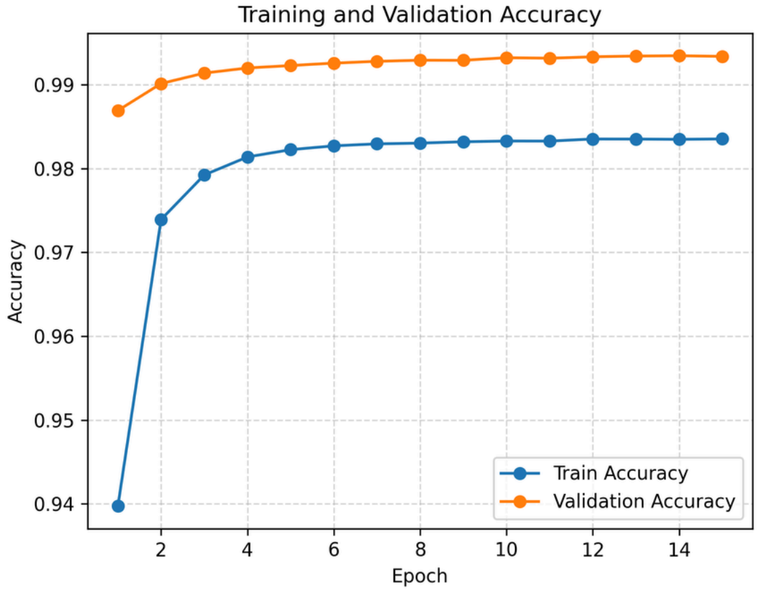
```python
# ================================
# Model Definition with EfficientNet-B7
# ================================

def get_model(pretrained: bool = True, num_classes: int = cfg.OUTPUTS_A) -> nn.Module:
    """
    Initialize and return the pre-trained EfficientNet-B7 model.
    """
    model = models.efficientnet_b7(weights=models.EfficientNet_B7_Weights.IMAGENET1K_V1 if pretrained else None)

    # Freeze all layers except the classifier
    for param in model.parameters():
        param.requires_grad = False

    # Replace the classifier head
    num_ftrs = model.classifier[1].in_features
    model.classifier = nn.Sequential(
        nn.Dropout(p=0.5, inplace=True),
        nn.Linear(num_ftrs, num_classes)
    )

    return model.to(cfg.DEVICE)
```

# MODEL ARCHITECTURE EFFICIENT NET B7

- **Hyperparameters:**
  - **Batch Size:** 128
  - **Learning Rate:** 0.0001
  - **Epochs:** 30
  - **Image Size:** 224x224 pixels
- **Activation Function:**
  - Uses Swish activation function as default in EfficientNet-B7.
- **Loss Function**:
  - Cross-Entropy Loss for multi-class classification.
- **Early Stopping:**
  - Stops training if validation loss does not improve after 5 epochs (patience = 5).
- **Learning Rate Scheduler:**
  - ReduceLROnPlateau: Reduces learning rate when validation loss plateaus.
- **Model Customization:**
  - Replaces EfficientNet-B7 classifier with a custom layer for 13 classes (chess pieces and empty grid).
- **Model Saving and Metrics Plotting:**
  - Saves the best model based on validation loss.
  - Plots training/validation loss and accuracy metrics.

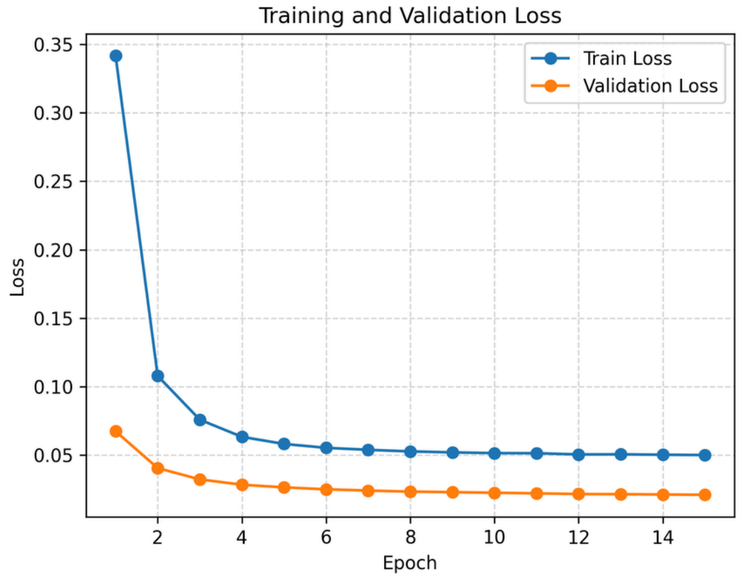
## Training and Validation Accuracy:

- **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.

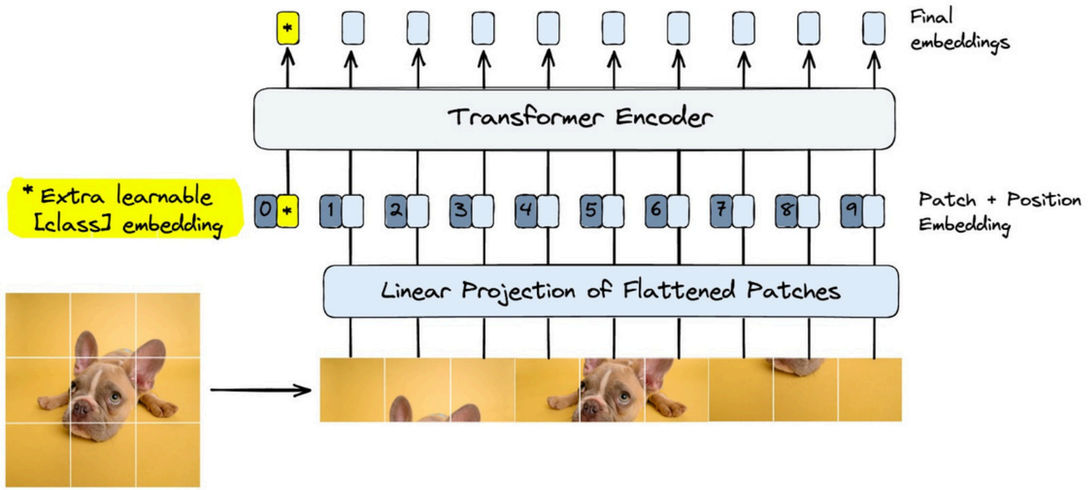



## Training and Validation Loss:

- **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.



# MODEL ARCHITECTURE
# VISION TRANSFORMER



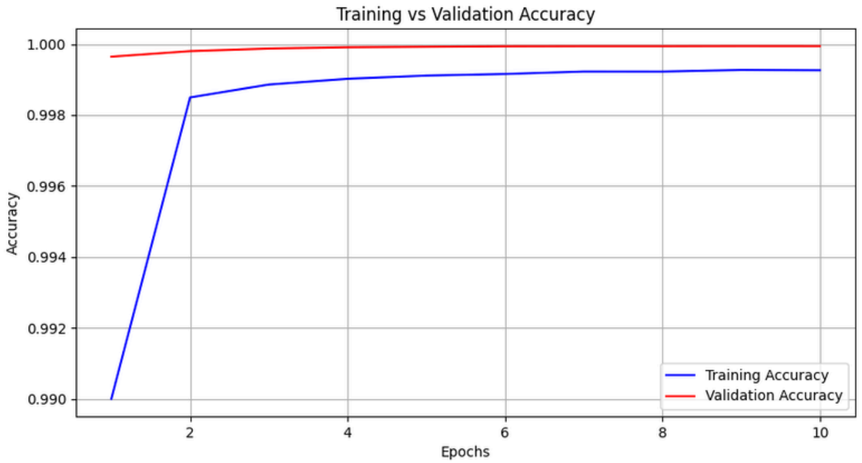

```python
def get_model(pretrained: bool = True, num_classes: int = LABEL_SIZE) -> nn.Module:
    """
    Initialize and return the pre-trained Vision Transformer (ViT) model.
    """
    # Import Vision Transformer from torchvision
    from torchvision.models import vit_b_16, ViT_B_16_Weights

    # Initialize ViT-B/16 model
    weights = ViT_B_16_Weights.IMAGENET1K_V1 if pretrained else None
    model = vit_b_16(weights=weights)

    # Freeze all layers except the classifier
    for param in model.parameters():
        param.requires_grad = False

    # Replace the classifier head
    num_ftrs = model.heads.head.in_features
    model.heads.head = nn.Linear(num_ftrs, num_classes)

    return model.to(DEVICE)
```

# MODEL ARCHITECTURE VISION TRANSFORMER

- **Hyperparameter Configuration:**
  - **Batch Size:** 256
  - **Learning Rate:** 0.0001
  - **Weight Decay:** 0.0001
  - **Epochs:** 10
  - **Scheduler:** CosineAnnealingLR, adjusts the learning rate according to a cosine schedule.
- **Loss Function:**
  - **Cross-Entropy Loss:** Utilized for multi-class classification, measures the performance of the model’s output against true labels.

## 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.

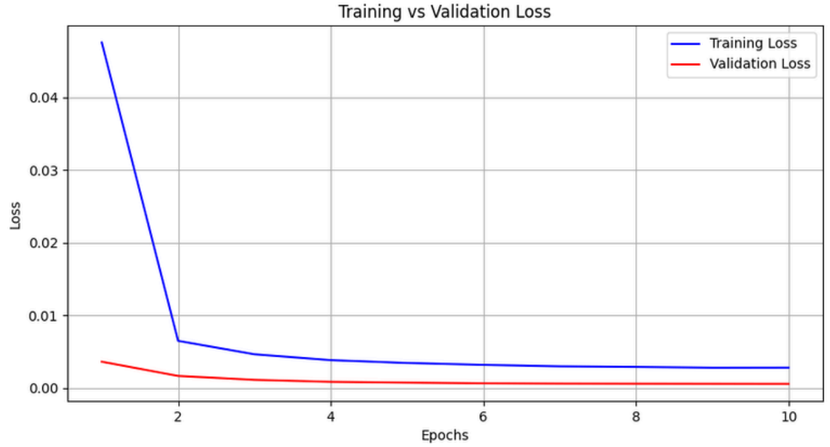



## Training vs Validation Loss:

**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.



# EXPLAINABILITY USING CAPTUM

**Objective:** Implemented Captum's Integrated Gradients to visualize and understand key influencers in model predictions.

**Key Benefits:**

- **Transparency:** Reveals which features most influence model decisions.
- **Trust and Reliability:** Enhances user and stakeholder trust by clarifying how decisions are made.
- **Effective Debugging:** Helps identify and correct biases or inaccuracies in feature processing.

**Implementation Highlights:**

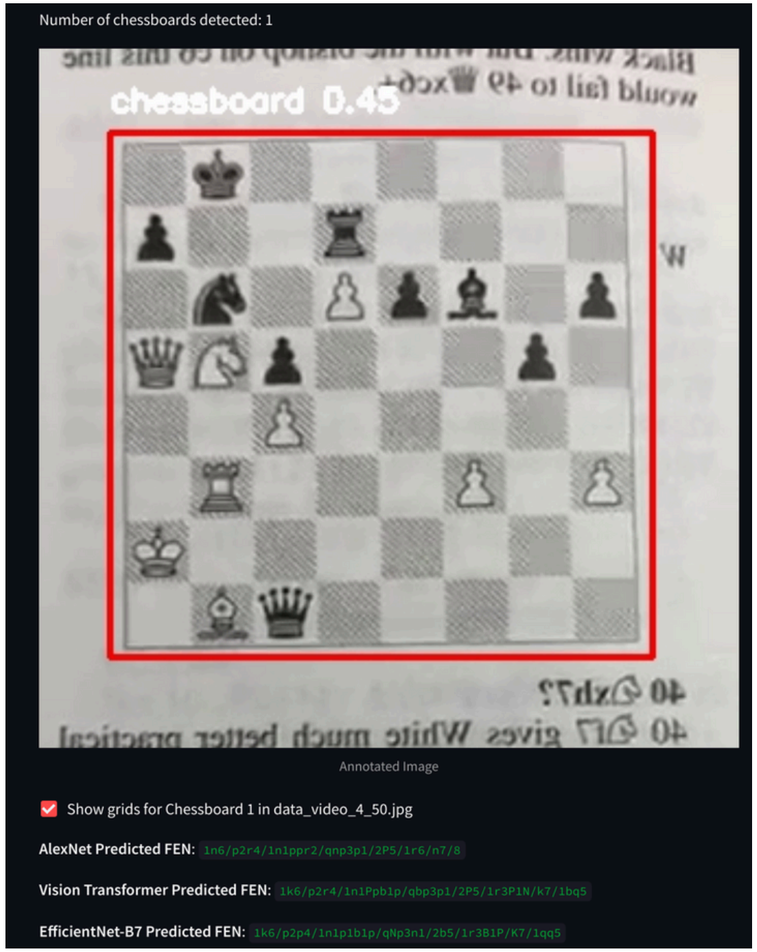
- **Attribution Visualization:** Uses gradient-based techniques to produce heatmaps showing critical regions influencing model outputs.
- **Iterative Improvement:** Enables model refinement by focusing on relevant input features, reducing the influence of irrelevant data.


# YOLO OBJECT DETECTION

- **Version: YOLOv7**
- **Technique : Corner Detection + Perspective Shift Pipeline**
  - **Adaptive Thresholding:** Applied using cv2.adaptiveThreshold() to enhance edges and regions of interest.
  - **Find Contours:** Utilized cv2.findContours() to identify contours in the thresholded image.
  - **Select Largest Contour:** Determined using cv2.contourArea() to focus on the primary object of interest.
  - **Find Corners:** Detected corners with cv2.arcLength() and approximated them with cv2.approxPolyDP().
  - P**erspective Shift:** Aligned detected corners using a fixed grid of 64 squares.

# STREAMLIT


Upload Image(s)
Drag and drop files here
Limit 200MB per file • JPG, JPEG, PNG, BMP, TIFF, TIF
Browse files
Screenshot from 2024-12-09 16-19-14.png 40.0KB
Processing Screenshot from 2024-12-09 16-19-14.png
Uploaded Image
Show grids for Chessboard 1 in Screenshot from 2024-12-09 16-19-14.png
AlexNet Predicted FEN: 8/8/8/2r5/2B2Q2/1N2n3/2k5/6k1
Vision Transformer Predicted FEN: 8/8/8/2r5/2B2Q2/1N2n3/2k5/6k1
EfficientNet-B7 Predicted FEN: 8/8/8/2r5/2B2Q2/1N2n3/2k5/6k1


Number of chessboards detected: 1
chessboard 0.45
Annotated Image
Show grids for Chessboard 1 in data_video_4_50.jpg
AlexNet Predicted FEN: 1n6/p2r4/1n1ppr2/qnp3p1/2P5/1r6/n7/8
Vision Transformer Predicted FEN: 1k6/p2r4/1n1Ppb1p/qbp3p1/2P5/1r3P1N/k7/1bq5
EfficientNet-B7 Predicted FEN: 1k6/p2p4/1n1p1b1p/qNp3n1/2b5/1r3B1P/K7/1qq5

# THANK YOU!