Chess Piece Classification Using Image Classification Algorithms

1. Objective

The goal of this project was to develop an image classification model to classify chess pieces into one of six classes: Bishop, King, Queen, Rook, Knight, and Pawn. This task involved training multiple image classification models and fine-tuning them to achieve the best possible performance.

2. Dataset

The dataset used in this project was provided as part of the assessment. It was downloaded from a Google Drive link, where I downloaded as a compressed archive. After extracting the dataset, I used it in my environment to train the image classification models. The dataset consists of labeled images of chess pieces, each belonging to one of the six classes mentioned above.

3. Approach

Data Preprocessing & Augmentation

To improve model performance and prevent overfitting, various data augmentation techniques were used. These techniques help generate more diverse training examples by applying random transformations to the images, such as rotation, shifts, and zoom. The following image augmentation strategy was applied:

```
# Define the data generator with augmentation
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    zoom_range=0.2
)

train_gen = datagen.flow(X_train, y_train, batch_size=32)
val_gen = ImageDataGenerator().flow(X_val, y_val, batch_size=32)
test_gen = ImageDataGenerator().flow(X_test, y_test, batch_size=32)
```

This augmentation was applied to the training data, while validation and testing data were left unaltered to ensure fair evaluation.

Fine-tuned Models

Three models were fine-tuned for this task:

1. Custom CNN Model

The first model is a custom Convolutional Neural Network (CNN) built from scratch with three convolutional layers, followed by fully connected layers. Dropout and batch normalization were applied to prevent overfitting and improve generalization.

```python
def create_model():
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)),
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Dropout(0.25),

        Conv2D(64, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Dropout(0.25),

        Conv2D(128, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Dropout(0.4),

        Flatten(),
        Dense(256, activation='relu'),
        BatchNormalization(),
        Dropout(0.5),
        Dense(len(class_names), activation='softmax')
    ])

    # Compile model
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

model = create_model()
```

2. ResNet50 Model

The second model used was ResNet50, a pre-trained model from the imagenet dataset. The ResNet50 model was used with the top layers removed, and custom layers were added on top for fine-tuning.

```python
base_model = tf.keras.applications.ResNet50(
    weights='imagenet', include_top=False, input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)
)

# Freeze the base model layers to prevent updates during initial training
base_model.trainable = False

# Add custom layers on top of the pretrained model
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(256, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
output = Dense(len(class_names), activation='softmax')(x)

# Create the final model
model = Model(inputs=base_model.input, outputs=output)

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

3. Inception V3 Model

The third model used was Inception V3, another pre-trained model, which was similarly modified with custom layers added for classification.

```python
from tensorflow.keras.applications import InceptionV3

# Load the InceptionV3 model with pretrained ImageNet weights
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(IMG_HEIGHT, IMG_WIDTH, 3))

# Freeze the base model's layers to retain learned features
base_model.trainable = False

# Build custom classifier on top of the base model
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
output = Dense(len(class_names), activation='softmax')(x)
```

For each of these models, **hyperparameter tuning** was conducted using Keras Tuner. However, the performance across all models was similar, achieving a maximum accuracy of around 0.45.

Final Model - Vision Transformer (ViT)

After experimenting with traditional CNN architectures, the performance was limited, leading to the exploration of **Vision Transformer (ViT)**. The ViT model, which leverages transformer-based self-attention mechanisms originally designed for NLP tasks, has demonstrated superior performance on vision tasks compared to conventional CNNs, particularly in scenarios where large datasets are available.

Training the Vision Transformer (ViT)

1. **Image Preprocessing**:
    - The images were preprocessed using ViTImageProcessor from Hugging Face's transformers library. The processor applies necessary transformations, including resizing and normalization, based on the mean and standard deviation of the ImageNet21k dataset. This normalization is critical for ensuring the images are in the same distribution as the data the model was pre-trained on.
    - **Training Transformations**: Data augmentation techniques such as random resized crops and horizontal flipping were used to increase the variability of the input data.
    - **Validation Transformations**: For validation, the images were resized to the target size and center-cropped to ensure consistent input dimensions for evaluation.
2. **Model Architecture**: The ViT model architecture used for this task was the google/vit-base-patch16-224-in21k model, which is based on the Vision Transformer with 16x16 patches and a base model size. The model was modified to classify the six chess pieces by adding a custom output layer. The pre-trained model's weights were frozen initially to prevent overfitting during fine-tuning, and only the top layers were trained.
3. **Training Setup**:
    - **TrainingArguments**: The ViT model was trained using the Hugging Face TrainingArguments API, which specifies important parameters like the learning rate, number of epochs, and batch sizes. The learning rate was set to 2e-5, and the model was trained for 30 epochs. Early stopping was applied with a patience of 5 epochs to prevent the model from overfitting.
    - **Metrics**: The performance was evaluated using several metrics, including accuracy, precision, recall, and F1 score. These metrics helped assess how well the model was performing in terms of both overall accuracy and its ability to handle class imbalances.

4. **Results**: The Vision Transformer achieved **significantly better performance** than the CNN and pre-trained models, with an accuracy of **0.85** on the validation set. Additionally, the F1 score, precision, and recall were also improved, indicating a better-balanced model that generalized well across all classes of chess pieces.

To further optimize performance, **early stopping** with a patience of 5 epochs was implemented to prevent overfitting.

Metrics and Evaluation

The model's performance was evaluated using a variety of metrics, including **accuracy**, **F1 score**, **precision**, and **recall**. These metrics help provide a comprehensive assessment of the model's ability to classify chess pieces, particularly in the context of any class imbalances that might exist.

Below is an overview of the key metrics and the confusion matrix that helped assess the model's classification performance.

Metrics Overview

Epoch	Validation Loss	Accuracy	F1 Score	Precision	Recall
1	1.672018	0.482143	0.386967	0.482143	0.413400
2	1.465169	0.642857	0.741369	0.642857	0.612647
3	1.174160	0.732143	0.779152	0.732143	0.721039
4	0.981252	0.839286	0.858866	0.839286	0.843313
5	0.831173	0.821429	0.816357	0.821429	0.816357
6	0.729118	0.910714	0.912167	0.917163	0.910714

Interpretation of the Metrics:

- **Accuracy**: This metric shows the proportion of correct predictions out of all predictions made. As shown, the model's accuracy steadily increased over the epochs, peaking at around **91%** in epoch 6.
- **F1 Score**: The F1 score is a balanced metric that considers both precision and recall. It is particularly useful when dealing with class imbalances. The F1 score improves consistently throughout the epochs, reaching **0.912** by the 6th epoch, indicating that the model is increasingly able to balance between precision and recall.
- **Precision**: Precision measures the proportion of true positive predictions out of all positive predictions made by the model. The precision steadily increased over the epochs and was **0.917** in the final epoch, suggesting the model is becoming better at correctly identifying the chess pieces when it predicts them as belonging to a specific class.
- **Recall**: Recall measures the proportion of true positive predictions out of all actual positive instances. The recall increased gradually, reaching **0.911** by epoch 6, showing that the model is becoming better at capturing all the positive instances (true chess pieces).

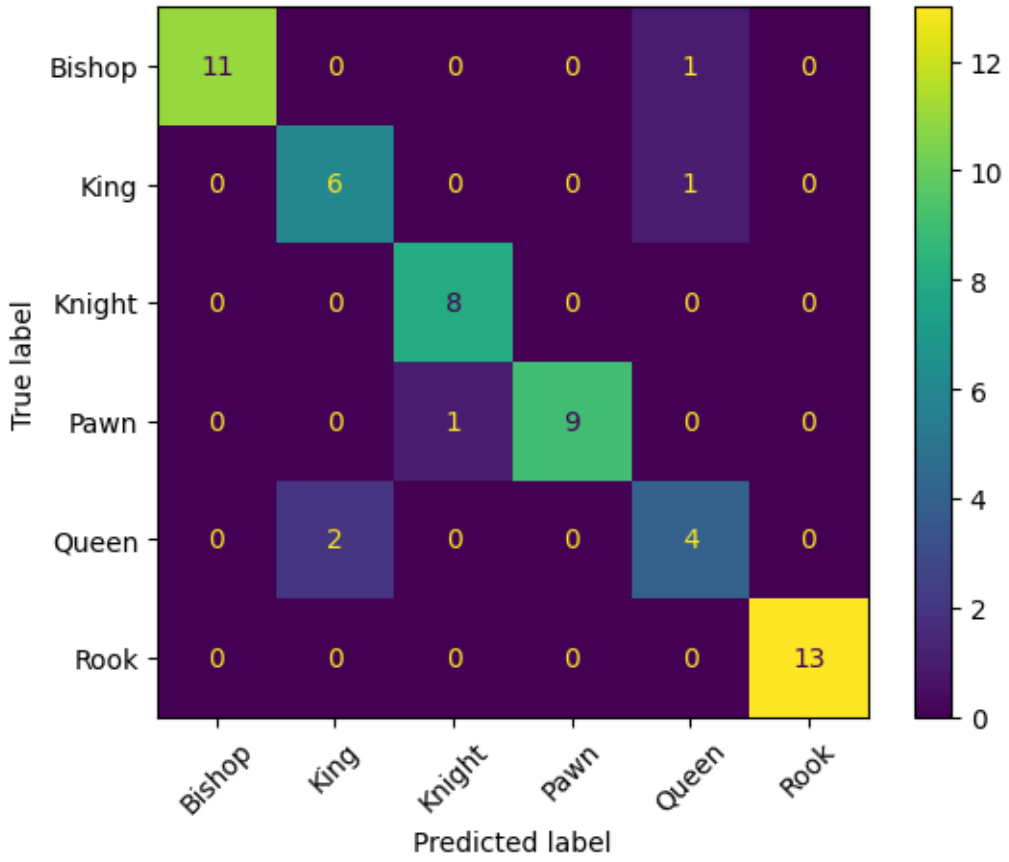
Test Performance

Metric	Value
Test Loss	0.7291
Test Accuracy	0.9107
Test F1 Score	0.9122

Test Precision	0.9172
Test Recall	0.9107

The test set performance is a strong indicator that the model is able to perform well on data it has never seen before, achieving an accuracy of **91.07%** on the test data. The F1 score of **0.9122** suggests that the model is performing well on all classes, and the precision and recall values indicate that it is effectively balancing between false positives and false negatives.

Confusion Matrix



4. Results and Performance

- **Best Model**: After experimenting with the CNN, ResNet50, and Inception V3 models, the Vision Transformer (ViT) model achieved the best performance.
- **Accuracy**: The Vision Transformer model significantly improved accuracy, surpassing previous models, with an accuracy around **0.85**.
- **Other Metrics**: The Vision Transformer also showed a high F1 score, precision, and recall, indicating a well-balanced model despite any class imbalances in the dataset.