CPE 019 - Final Project - Training and Saving the Model

Names: Quejado, Jimlord M., Serrano, Jio A.
Course and Section: CPE019 - CPE32S3
Instructor: Engr. Roman Richard

Model Training

This Colaboratory Notebook contains the training of the model as well as the saving of the model in .H5 format to be deployed in Streamlit.

About the data

The data is composed of various images of chess pieces including the bishop, king, knight, pawn, queen, and rook. The dataset is composed of 76-107 images each class.

Associated Tasks

- Classification

Feature Type

- Image

Instances

- 556

Classes

- 6

Link to data set:

https://www.kaggle.com/datasets/niteshfre/chessman-image-dataset

Preparing the Data

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [2]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Splitting the Dataset

In [3]:
```
!pip install split-folders
```

```
Collecting split-folders
  Downloading split_folders-0.5.1-py3-none-any.whl (8.4 kB)
Installing collected packages: split-folders
Successfully installed split-folders-0.5.1
```

In [4]:
```
import splitfolders
import os

path = "/content/drive/MyDrive/Chessman-image-dataset"
print(os.listdir(path))
```

```
['King', 'Queen', 'Bishop', 'Pawn', 'Knight', 'Rook']
```

In [5]:

```
# Splitting the dataset into training, testing, and validation images

splitfolders.ratio(path, seed=25, output="Chess-Splitted3", ratio=(0.2, 0.2, 0.6))
```

```
Copying files: 556 files [02:29,  3.72 files/s]
```

Loading the Images

In [6]:

```
size = 224
```

In [7]:

```
import matplotlib.pyplot as plt
import os
import numpy as np
import tensorflow as tf
import pandas as pd
import random
import splitfolders
import cv2
import glob
import csv
from tensorflow.keras.preprocessing import image
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers.experimental import preprocessing
import seaborn as sns
from sklearn.metrics import classification_report
```

In [8]:

```
import random

images = []
class_names = ["Bishop", "King", "Knight", "Pawn", "Queen", "Rook"]
og_image_dir = "/content/drive/MyDrive/Chessman-image-dataset"

for class_name in class_names:
    image_files = os.listdir(os.path.join(og_image_dir, class_name))
    image_file = random.choice(image_files)
    images.append(os.path.join(og_image_dir, class_name, image_file))

fig, ax = plt.subplots(2, 3, figsize = (9, 6))

for i in range(6):
    ax[i // 3, i % 3].imshow(plt.imread(images[i]))
    ax[i // 3, i % 3].set(title = class_names[i])
    ax[i//3, i%3].axis('off');
```

Remarks: The images are not yet of the same sizes, hence, we will be needing to perform image pre-processing before training our model.

Image Preprocessing

Resizing the Images

In this section we will be resizing the images into 224 by 224 files.

In [11]:
```
split_image_dir = "/content/Chess-Splitted3"
```

In [12]:
```
main_directories = os.listdir(split_image_dir)

for main_directory in main_directories:
    main_directory_path = os.path.join(split_image_dir, main_directory)
    sub_directories = os.listdir(main_directory_path)

    for sub_directory in sub_directories:
        sub_directory_path = os.path.join(main_directory_path, sub_directory)
        image_paths = glob.glob(os.path.join(sub_directory_path, '*jpg'))

        for image_path in image_paths:
            img = cv2.imread(image_path)
            img = cv2.resize(img, (size, size), interpolation = cv2.INTER_CUBIC)
            img = img.astype(np.float32)
            cv2.imwrite(image_path, img)
```

Loading Datasets

Here, we will load the datasets that we have created earlier—train, test, and validation. This will help us verify the number of files belonging to each of the datasets created. When deploying our model in Streamlit, it is important to take note of its size, since we will be using GitHub, the file size of the model should not be greater than 25 MB. Hence, we decrease the size of our training dataset.

In [13]:
```
# Load the training dataset
train_data = tf.keras.preprocessing.image_dataset_from_directory("/content/Chess-Splitted3/train",
                                                                  label_mode = "categorical",
                                                                  image_size = (size, size) ,
                                                                  seed = 42)
```

```
Found 109 files belonging to 6 classes.
```

In [14]:
```
# Load the testing dataset
test_data = tf.keras.preprocessing.image_dataset_from_directory("/content/Chess-Splitted3/test",
                                                                  label_mode = "categorical",
                                                                  image_size = (size, size) ,
                                                                  seed = 42)
```

```
Found 334 files belonging to 6 classes.
```

In [15]:
```
# Load the validation dataset
val_data = tf.keras.preprocessing.image_dataset_from_directory("/content/Chess-Splitted3/val",
                                                                  label_mode = "categorical",
                                                                  image_size = (size, size) ,
                                                                  seed = 42)
```

```
Found 109 files belonging to 6 classes.
```

Performing Image Augmentation

In this section, image augmentation will be performed. This allows us to increase the size of our dataset without the need to add new images. In this step, existing images will be augmented by adding different parameters like tilts and shifts. Performing this makes our model more robust to changes to the images fed to it [1].

---

[1] Data augmentation, “Data augmentation,” TensorFlow, 2024. https://www.tensorflow.org/tutorials/images/data_augmentation (accessed May 18, 2024).

In [16]:
```
# Image Augmentation layer
image_augmentation = Sequential([
  preprocessing.RandomFlip('horizontal', seed = 42),
  preprocessing.RandomRotation(0.2, seed = 42),
  preprocessing.RandomHeight(0.2, seed = 42),
  preprocessing.RandomWidth(0.2, seed = 42),
  preprocessing.RandomZoom(0.2, seed = 42)])
```

Callbacks

In this section, we will be defining where our model checkpoints will be saved and the criteria that will be followed for saving.

First, a callback that will save the best model based on validation accuracy during training is defined. Second, we define a callback to reduce the learning rate when validation accuracy is not improving. And lastly, we define a callback to log our training history into a CSV file.

In [17]:
```
checkpoint_dir = "/content/drive/MyDrive/Final Project Checkpoints"
```

```python
# Callback to save the best model based on validation accuracy during training
model_checkpoint = tf.keras.callbacks.ModelCheckpoint(checkpoint_dir,
                                                      save_best_only = True,
                                                      monitor = "val_accuracy")

# Callback to reduce the learning rate when the validation accuracy stops improving
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor = "val_accuracy",
                                                 factor = 0.2,
                                                 patience = 3,
                                                 min_lr = 1e-7)

# Callback to log training history to a csv file
base_csv_logger = tf.keras.callbacks.CSVLogger('/content/drive/MyDrive/Final Project Checkpoints/CSV Files/Base
ft_csv_logger = tf.keras.callbacks.CSVLogger('/content/drive/MyDrive/Final Project Checkpoints/CSV Files/Fine-Tu
```

Creating a Base Line Model

In [18]:

```python
base_model = tf.keras.applications.VGG19(include_top = False)
base_model.trainable = False

inputs = tf.keras.Input(shape = (size, size, 3))

# Applying Image Augmentation
x = image_augmentation(inputs)

# Normalizing Image
x = layers.Rescaling(1./255)(x)

x = base_model(inputs)
x = layers.Dropout(0.4)(x)
x = layers.Conv2D(256, 3, activation = 'relu', padding = 'same')(x)
x = layers.Dropout(0.5)(x)

x = tf.keras.layers.GlobalMaxPooling2D()(x)
x = layers.Dropout(0.6)(x)

x = layers.Dense(6)(x) # Output layer
outputs = layers.Activation("softmax")(x)

# Create the model
model = tf.keras.Model(inputs, outputs)

# Compile the model
model.compile(loss = "categorical_crossentropy",
              optimizer = tf.keras.optimizers.Adam(),
              metrics = ["accuracy"])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_or
dering_tf_kernels_notop.h5
80134624/80134624 [==============================] - 1s 0us/step
```

In [19]:

```python
model.compile(loss = "categorical_crossentropy",
              optimizer = tf.keras.optimizers.Adam(),
              metrics = ["accuracy"])
```

Training the Model

In [21]:

```python
# Training the base model
model.fit(train_data,
          epochs = 10,
          steps_per_epoch = len(train_data),
          validation_data = val_data,
          validation_steps = len(val_data),
          callbacks = [model_checkpoint, reduce_lr, base_csv_logger])
```

```
Epoch 1/10
4/4 [==============================] - 228s 69s/step - loss: 119.3006 - accuracy: 0.2569 - val_loss: 21.9210 - v
al_accuracy: 0.3211 - lr: 0.0010
Epoch 2/10
4/4 [==============================] - 163s 47s/step - loss: 103.5002 - accuracy: 0.2844 - val_loss: 12.1766 - v
al_accuracy: 0.4954 - lr: 0.0010
Epoch 3/10
4/4 [==============================] - 165s 48s/step - loss: 52.3187 - accuracy: 0.4954 - val_loss: 12.9506 - va
l_accuracy: 0.5046 - lr: 0.0010
Epoch 4/10
4/4 [==============================] - 161s 46s/step - loss: 61.7797 - accuracy: 0.4771 - val_loss: 10.0215 - va
l_accuracy: 0.5963 - lr: 0.0010
Epoch 5/10
4/4 [==============================] - 163s 46s/step - loss: 35.9299 - accuracy: 0.6422 - val_loss: 11.0192 - va
l_accuracy: 0.6239 - lr: 0.0010
Epoch 6/10
4/4 [==============================] - 163s 47s/step - loss: 31.1447 - accuracy: 0.6606 - val_loss: 10.8770 - va
l_accuracy: 0.6330 - lr: 0.0010
Epoch 7/10
4/4 [==============================] - 161s 47s/step - loss: 26.0128 - accuracy: 0.7431 - val_loss: 9.6392 - val
_accuracy: 0.6972 - lr: 0.0010
Epoch 8/10
4/4 [==============================] - 161s 46s/step - loss: 17.0424 - accuracy: 0.7339 - val_loss: 8.7101 - val
_accuracy: 0.6972 - lr: 0.0010
Epoch 9/10
4/4 [==============================] - 161s 46s/step - loss: 18.3325 - accuracy: 0.7982 - val_loss: 7.6897 - val
_accuracy: 0.6972 - lr: 0.0010
Epoch 10/10
4/4 [==============================] - 155s 44s/step - loss: 10.5751 - accuracy: 0.8165 - val_loss: 8.3951 - val
_accuracy: 0.6697 - lr: 0.0010
```

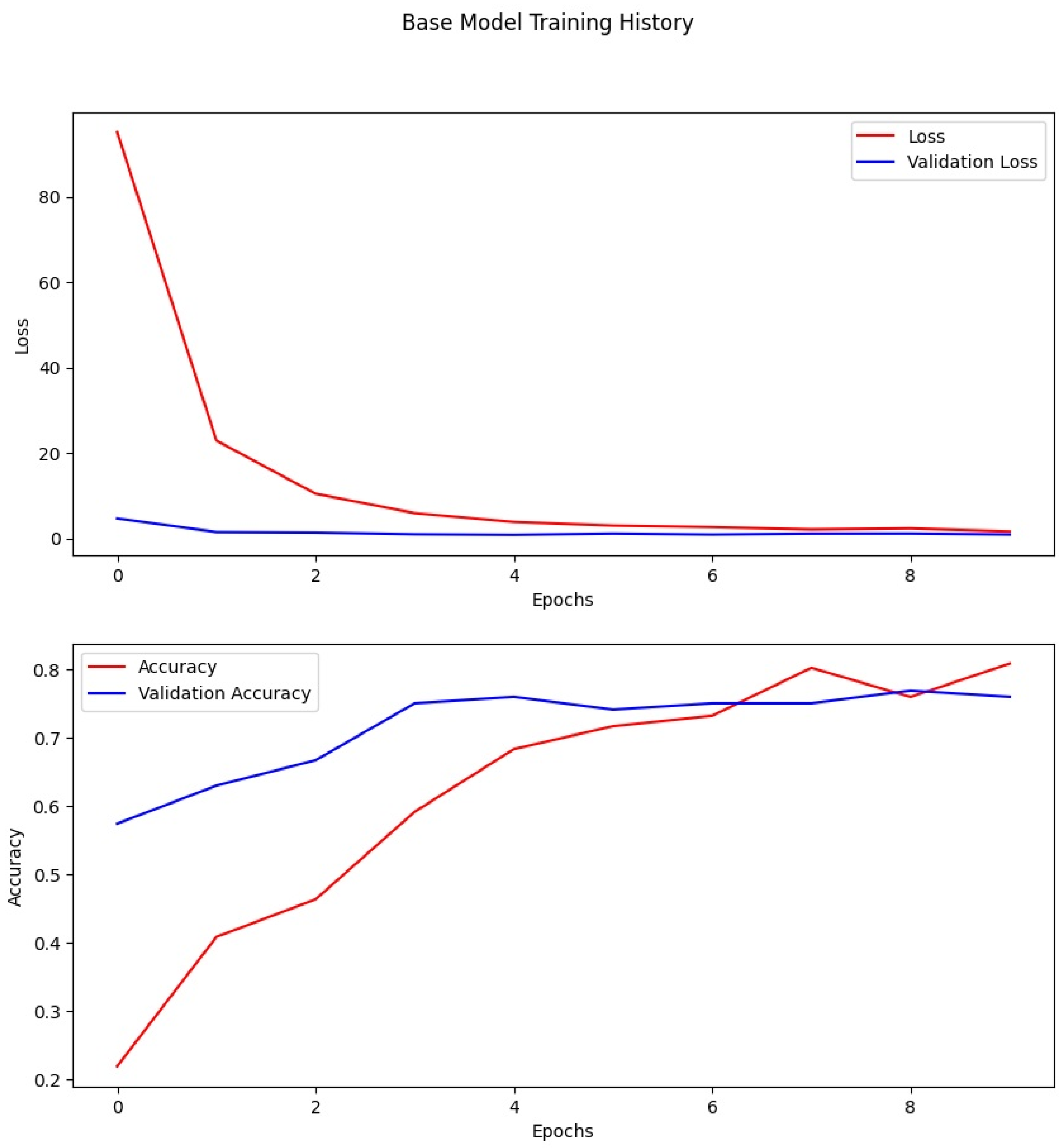
Out[21]:

```
<keras.src.callbacks.History at 0x79256f5cfb20>
```

Visualizing Model Performance

In [30]:

```python
# Load the base model's training history
history = pd.read_csv('/content/drive/MyDrive/Final Project Checkpoints/CSV Files/Base Model History.csv')

# Extract the values for loss, val_loss, accuracy, and val_accuracy
loss = history['loss']
val_loss = history['val_loss']
accuracy = history['accuracy']
val_accuracy = history['val_accuracy']

# Define number of epochs
epochs = range(len(loss))

# Plot the training history using two subplots (for loss and accuracy)
fig, ax = plt.subplots(2, 1, figsize = (10, 10))

fig.suptitle("\nBase Model Training History")

ax[0].plot(epochs, loss, label = 'Loss', color = "r")
ax[0].plot(epochs, val_loss, label = 'Validation Loss', color = "b")
ax[0].set_ylabel('Loss')
ax[0].set_xlabel('Epochs')
ax[0].legend()

ax[1].plot(epochs, accuracy, label = 'Accuracy', color = "r")
ax[1].plot(epochs, val_accuracy, label = 'Validation Accuracy', color = "b")
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Accuracy')
ax[1].legend();
```



Saving the Model

In [23]:
```
model.save('/content/drive/MyDrive/Final Project Checkpoints/base_model2.h5')
print('Model successfully saved to disk.')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your mode
l as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the nati
ve Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
Model successfully saved to disk.
```

Finalized model

In [24]:
```
# Unfreezing the base model entirely
base_model.trainable = True

# Freezing all layers in the base model, except the last 10 layers
for layer in base_model.layers[:-10]:
  layer.trainable = False

# Compiling the model
model.compile(loss = "categorical_crossentropy",
              optimizer = tf.keras.optimizers.Adam(learning_rate = 2.0000e-05),
              metrics = ["accuracy"])
```

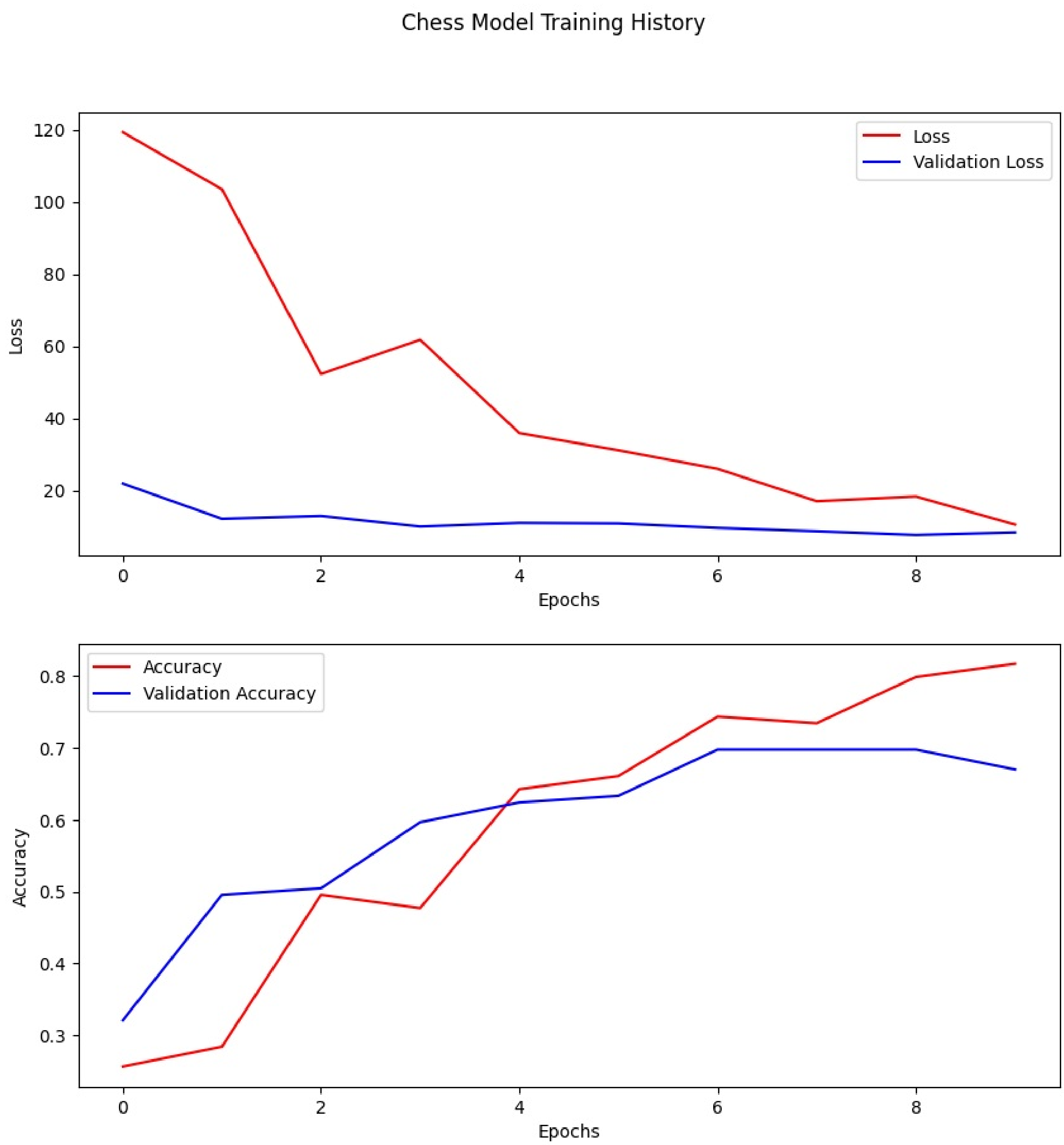
In [25]:
```
# Training the fine-tuned model
model.fit(train_data,
          epochs = 10,
          steps_per_epoch = len(train_data),
          validation_data = val_data,
          validation_steps = len(val_data),
          callbacks = [model_checkpoint, ft_csv_logger, reduce_lr])
```

```
Epoch 1/10
4/4 [==============================] - 240s 64s/step - loss: 6.0473 - accuracy: 0.8899 - val_loss: 8.3114 - val_
accuracy: 0.7064 - lr: 2.0000e-05
Epoch 2/10
4/4 [==============================] - 238s 63s/step - loss: 4.6316 - accuracy: 0.8807 - val_loss: 7.4547 - val_
accuracy: 0.6972 - lr: 2.0000e-05
Epoch 3/10
4/4 [==============================] - 247s 67s/step - loss: 7.9496 - accuracy: 0.8532 - val_loss: 6.2805 - val_
accuracy: 0.7248 - lr: 2.0000e-05
Epoch 4/10
4/4 [==============================] - 236s 64s/step - loss: 7.8348 - accuracy: 0.8624 - val_loss: 5.6004 - val_
accuracy: 0.7248 - lr: 2.0000e-05
Epoch 5/10
4/4 [==============================] - 238s 63s/step - loss: 5.6457 - accuracy: 0.8716 - val_loss: 4.8235 - val_
accuracy: 0.7615 - lr: 2.0000e-05
Epoch 6/10
4/4 [==============================] - 237s 64s/step - loss: 3.8785 - accuracy: 0.8899 - val_loss: 4.3609 - val_
accuracy: 0.7523 - lr: 2.0000e-05
Epoch 7/10
4/4 [==============================] - 234s 63s/step - loss: 4.4738 - accuracy: 0.8624 - val_loss: 4.3881 - val_
accuracy: 0.7248 - lr: 2.0000e-05
Epoch 8/10
4/4 [==============================] - 237s 64s/step - loss: 3.1136 - accuracy: 0.8899 - val_loss: 4.5264 - val_
accuracy: 0.7064 - lr: 2.0000e-05
Epoch 9/10
4/4 [==============================] - 234s 63s/step - loss: 3.1563 - accuracy: 0.8807 - val_loss: 4.5004 - val_
accuracy: 0.7064 - lr: 4.0000e-06
Epoch 10/10
4/4 [==============================] - 238s 64s/step - loss: 3.6535 - accuracy: 0.8991 - val_loss: 4.4139 - val_
accuracy: 0.6972 - lr: 4.0000e-06
```

Out[25]:
```
<keras.src.callbacks.History at 0x79256012ddb0>
```

Visualizing Model Performance

In [32]:
```
# Load the base model's training history
history = pd.read_csv('/content/drive/MyDrive/Final Project Checkpoints/CSV Files/Base Model History 2.csv')

# Extract the values for loss, val_loss, accuracy, and val_accuracy
loss = history['loss']
val_loss = history['val_loss']
accuracy = history['accuracy']
val_accuracy = history['val_accuracy']

# Define number of epochs
epochs = range(len(loss))

# Plot the training history using two subplots (for loss and accuracy)
fig, ax = plt.subplots(2, 1, figsize = (10, 10))

fig.suptitle("\nChess Model Training History")

ax[0].plot(epochs, loss, label = 'Loss', color = "r")
ax[0].plot(epochs, val_loss, label = 'Validation Loss', color = "b")
ax[0].set_ylabel('Loss')
ax[0].set_xlabel('Epochs')
ax[0].legend()

ax[1].plot(epochs, accuracy, label = 'Accuracy', color = "r")
ax[1].plot(epochs, val_accuracy, label = 'Validation Accuracy', color = "b")
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Accuracy')
ax[1].legend();
```



Saving the Model

In [27]:

```
model.save('/content/drive/MyDrive/Final Project Checkpoints/chess_model.h5')
print('Model successfully saved to disk.')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your mode
l as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the nati
ve Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
Model successfully saved to disk.
```

Remarks: The model that will be used for Streamlit deployment will be the `chess_model.h5` which garnered a 76.21% best validation accuracy based on its history.