Department of Electrical Engineering

San Jose State University

EE 198B Senior Project Final Report

The DeepFakeBusters

Joseph Esteves (0153233750)

Abraham Parker (015997566)

Kendrick Tran (01306411)

Ching-Tang Chang (015672358)

Date

December 9, 2024

Project advisor: Professor Chang Choo

Note: Certain portions of this report were removed so that only the technical information remained. Edits were also made to make the report easier and faster to read

Overview

Our senior project group designed a Convolutional Neural Network (CNN) that we call "The Deepfake Buster" that does a binary classification on an image to determine whether it is a deepfake or not. A deepfake is an image that was either generated by artificial intelligence (AI) or modified by AI. After training our CNN for 47 epochs we got a validation and test accuracy of 96.88% and 91.07% respectively.

Methodology

The design of our model consists of the data collection and preparation, CNN architecture, and training of the model.

Dataset

- Training dataset has 140,002 images.
- Validation dataset has 10,905 images.
- Testing dataset has 39,428 images.
- Image size was 256 x 256.

All the datasets have the same amount of real and deepfake images, the images were already in subfolders labeled as real or fake when we downloaded them from Kaggle: https://www.kaggle.com/datasets/manjilkarki/deepfake-and-real-images?resource=download. [4] The original source of these images downloaded from kaggle is the OpenForensics: Multi-Face Forgery Detection And Segmentation In-The-Wild Dataset [V.1.0.0] available on Zenodo: https://zenodo.org/records/5528418#.YpdlS2hBzDd. [5]

OpenForensics Dataset Preparation and Augmentation

The following information about the OpenForensics dataset comes from [6].

How the deepfakes were generated.

OpenForesniscs used GAN models and Poisson blending to generate new faces from raw images collected from Google Open Images. The images collected were screened in that images containing no human faces were moved and images containing artificial depiction of human faces were also removed such as currency, art, and dolls and any unreal depictions of human faces such as cartoons were removed.

Data Augmentation

Data augmentation was done to allow us to better evaluate the performance of the model including:

- Color manipulation.
- Edge manipulation.
- Block-wise distortion.
- Image corruption.
- Convolution mask transformation.
- External effect.

Other Notable Features of the Dataset

- 64.7% of the images are indoor scenes, the rest outdoor scenes.
- 50:50 ratio of male to female faces.
- An average of 2.9 faces per image.

Training

The training for this model was for 100 epochs and the model was saved at different epochs throughout the training process allowing us to pick the model with the best validation and test accuracy. The 90x90history.json file was also updated at every epoch giving us the performance results for every epoch throughout the training history. This also allowed us to see which epoch the model that was saved is correspond with.

Training Parameters

The parameters were used for the training of our model throughout the entire 100 epochs.

- Images resized to 90 x 90.
- Batch size is 64.
- Adam Optimizer was used and set to 1×10^{-3}.
- Binary Cross-Entropy was used as the loss function.

$$Binary\ Cross - Entropy = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i\, log(p_i) + (1 - y_i)\, log(1 - p_i)\right]$$

Equation 1

In equation 1 y_i is the label of the data element in this case true or false. p_i is the predicted probability our model has that element is a certain label. N is the total number of data points.

Training Procedure

The SJSU College of Engineering HPC was used to train the model and the specifications and type of CPU can be seen in figure 4. During a training session, the code saves the model of the epoch with the best performance on a keras file named 90x90Best_.keras and the model of the most recent epoch on a file called 90x90Epoch.keras.

Due to the HPC's tendency to time out, the entire 100 epochs was unable to be done in a single training session. Therefore multiple keras files were made to record the best model during each training session. A total of 9 keras files were made to save the model with the best validation accuracy at different training sessions. The description and names of these files can be found in table 1 of the results section. The purpose of having these keras files was that it gives us the ability to try different training parameters starting from a particular epoch. For example if we found that we have overfitted after a certain epoch we can go back to a previous epoch and try different parameters. The 90x90Epoch.keras was constantly updated after every epoch and at the beginning of a training session the 90x90Epoch.keras was the model that was loaded into the system.

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                56
On-line CPU(s) list:   0-55
Thread(s) per core:    2
Core(s) per socket:    14
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 79
Model name:            Intel(R) Xeon(R) CPU E5-2660 v4 @ 2.00GHz
Stepping:              1
CPU MHz:               3095.703
CPU max MHz:           3200.0000
CPU min MHz:           1200.0000
BogoMIPS:              3991.10
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              35840K
NUMA node0 CPU(s):     0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46,48,50,52,54
NUMA node1 CPU(s):     1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45,47,49,51,53,55
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr
sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology
nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr
pdcm pcid dca sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm
3dnowprefetch epb cat_l3 cdp_l3 intel_pt ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust
bmi1 hle avx2 smep bmi2 erms invpcid rtm cqm rdt_a rdseed adx smap xsaveopt cqm_llc cqm_occup_llc
cqm_mbm_total cqm_mbm_local dtherm ida arat pln pts md_clear spec_ctrl intel_stibp flush_l1d
```

Figure 4 shows the type and specifications of the CPU used in the SJSU College of Engineering HPC.

CNN Architecture

In order to detect a deep fake image a CNN architecture is used. This architecture starts by taking a 90x90 sized image and using data augmentation techniques to increase the amount of

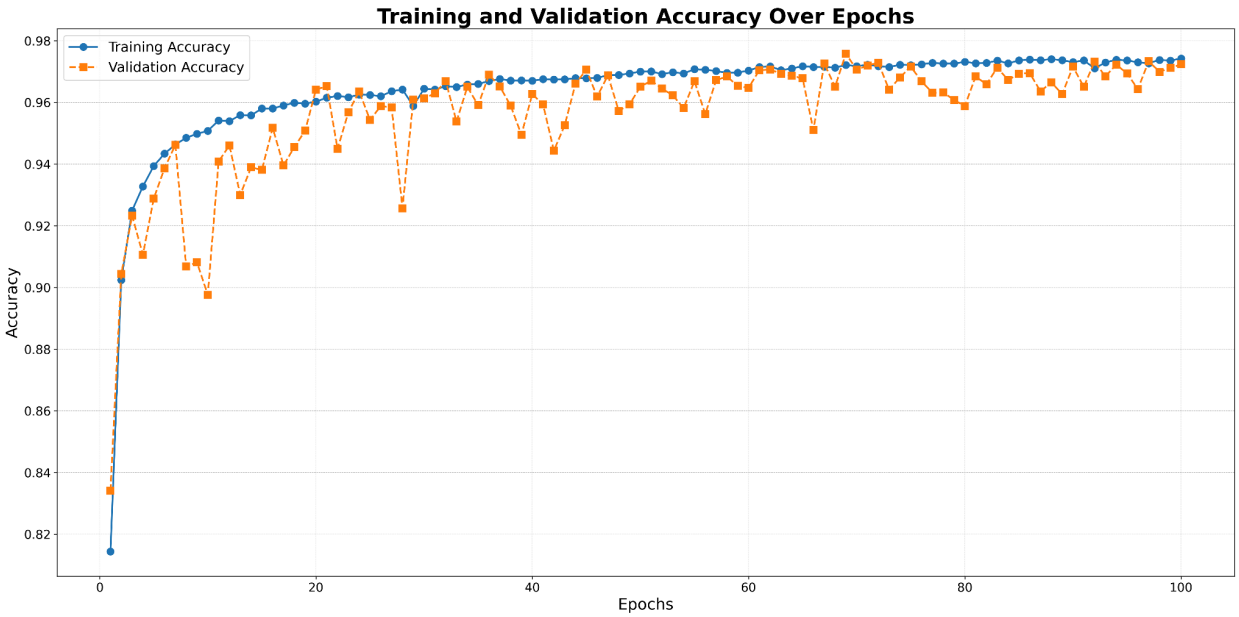
training data. These techniques include flipping the picture horizontally and vertically, rotating the picture in a random direction, and changing the brightness and contrast of the picture. After the picture is augmented the training data is passed through 6 convolutional layers.

Each convolutional layer is followed by a batch normalization layer to normalize the activation of the internal layers in order to make the model better at generalizing data. the data which increases accuracy. The first two convolutional layers have 32 filters with the filter size being 3x3. After that a max pooling layer with a 2x2 filter is used to reduce the spatial dimensions. This preserves key features and reduces computational complexity. Then a drop out layer is used to remove nodes from the CNN architecture in order to increase its ability to generalize data. The next two convolutional layers have 64 filters with the filter size being 3x3. The second layer is followed by another max pooling layer with a 2x2 filter which is then followed by another dropout layer. The final two convolutional layers have 128 filters with the filter size being 3x3. The last layer is followed by another max pooling layer with a 2x2 filter which is then followed by another dropout layer. After that a flatten layer is used to turn the data into a 2-dimensional array for the connected layers. Then 2 dense layers, with the first one being 128 nodes and the second one being 2 nodes, use the ReLU and softmax activation function respectively before the output is determined.

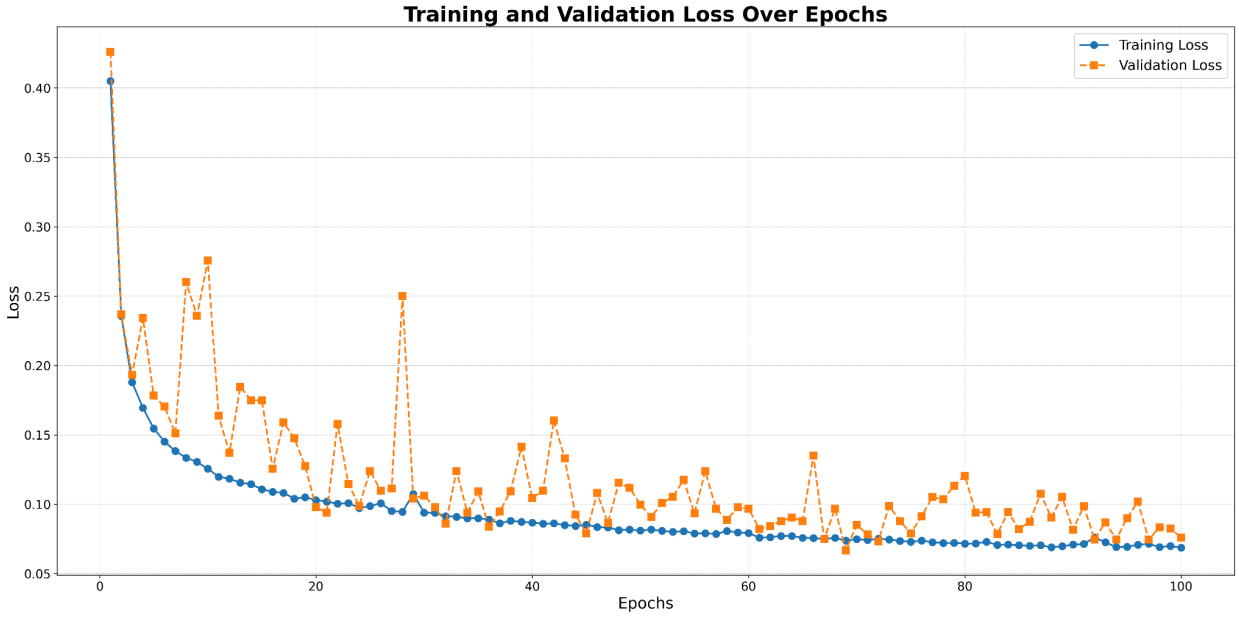
Results

File Name	Epoch	Validation	Test
90x90Best.keras	32	96.69%	88.13%
90x90Best1.keras	36	96.98%	90.44%
90x90Best2.keras	45	97.06%	89.09%
90x90Best3.keras	47	96.88%	91.07%
90x90Best4.keras	61	97.04%	89.21%
90x90Best5.keras	67	97.26%	89.54%
90x90Best6.keras	69	97.58%	89.63%
90x90Best7.keras	83	97.13%	87.53%
90x90Best8.keras	97	97.24%	89.65%
90x90Epoch.keras	100	97.24%	89.40%

Table 1 shows the file names of the saved models as well as the epoch that the model was created at and its accuracies for the validation and test dataset.



Graph 1 shows the accuracies of the model with the training and validation datasets for the first 100 epochs of the model's training.



Graph 2 shows the loss of the model with the training and validation datasets for the first 100 epochs of the model's training.

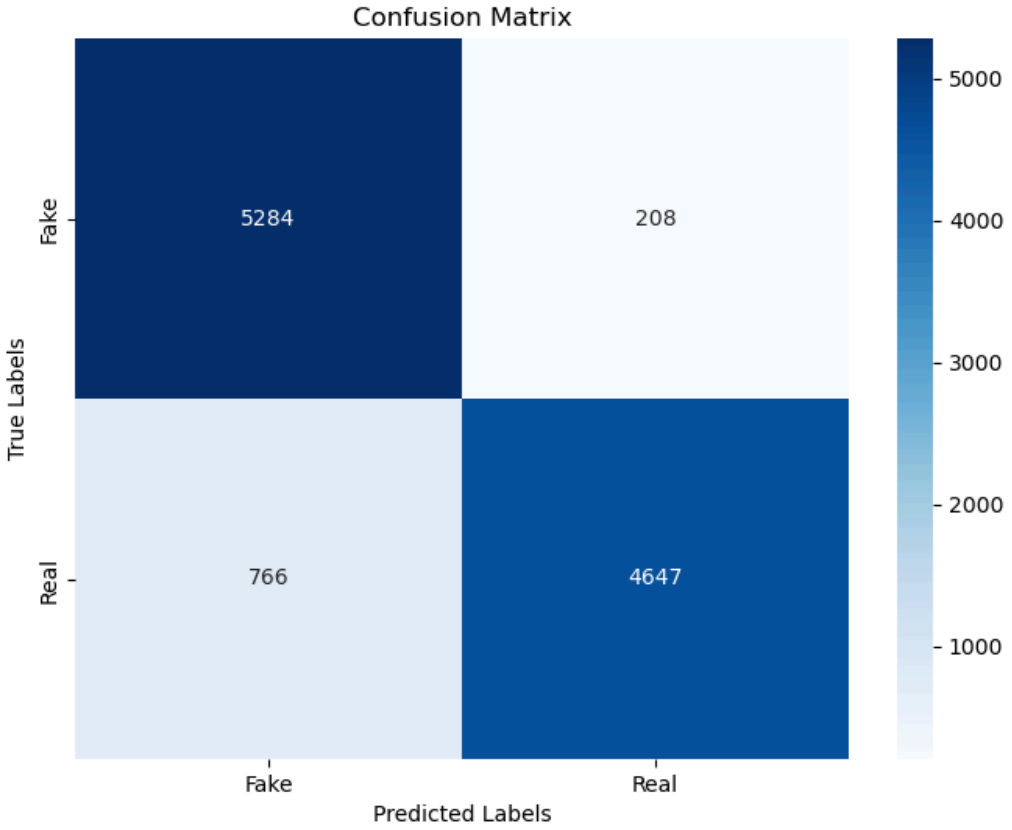


Figure 5 shows the confusion matrix for when our model in the 90x90Best3.keras file runs through the test dataset in which the darker squares refer to the amount of correct predictions made and the lighter squares represent the incorrect predictions.

Discussion

From table 1, we can see that the epoch with the best performance results when taking into account both the validation and testing accuracies is epoch 47 which is file 90x90Best3.keras having a validation accuracy of 96.88% and test accuracy of 91.07%. Looking at graph 1, the validation accuracy hovers around 97% after epoch 47 and the oscillations of the validation curve stabilize after the epoch 80. From the trend of the curve in graph 1, it doesn't appear that the training and validation accuracies will go above the 97% range if we keep all the parameters the same. The training and validation loss curve in graph 2 stabilizes at around a value of 0.07 and doesn't show any signs of significantly decreasing with further training. The confusion matrix in figure 5 shows that with using the model in the file 90x90Best3.keras on the test

dataset, out of the 6050 images that it predicted as fake, 0.127% of those images were real and out of the 4855 images that it predicted as real, 0.043% of those images were fake. Showing that the model is more likely to misclassify a fake image as real and performed better at confirming that images were real.

Conclusion

We were able to accomplish a good balance between the validation and test accuracy with our model on the dataset with a 91.07% test accuracy and 96.88% validation accuracy. The misclassification rates were low with 0.127% chance of misidentifying a fake image as real and a 0.043% chance of misidentifying a real image as fake. Furthermore we were able to provide a good starting point where we and others can build off of this model and what was recorded about it. With the training history recorded on the 90x90history.json file, we can see the performance of the model at any epoch in the training history and evaluate where we can move from different epochs. From the training history, we found that the model was overfitting after epoch 47 because the validation accuracy continued to hover around 97% and the test accuracy began to decrease. Therefore further work can be done on the model in 90x90Best3.keras with changing the hyperparameters to reduce generalization and/or training on a different dataset. Using different datasets to train the epoch 47 model seems to be the most significant way we can improve performance and make sure that the model works well with deepfakes generated by other algorithms other than GAN. Generating the deepfakes with more sophisticated algorithms to further train our model can be the way to ensure that our model operates well in the “real world” and keeps up with advancing deepfake technology.

Works Cited

[1] Spring, Marianna. “Trump Supporters Target Black Voters with Faked AI Images.” BBC News, BBC, 4 Mar. 2024, www.bbc.com/news/world-us-canada-68440150?zephr-modal-register.

[2] “Rising tide of deepfake pornography targets K-pop stars,” The Straits Times, Sep. 12, 2024. https://www.straitstimes.com/asia/east-asia/rising-tide-of-deepfake-pornography-targets-k-pop-stars

[3] Haroon Bhorat, Landry Signé, et al. “The Threat Posed by Deepfakes to Marginalized Communities.” Brookings, Brookings, 21 Apr. 2021, www.brookings.edu/articles/the-threat-posed-by-deepfakes-to-marginalized-communities/.

[4] M. Karju, Deepfake and Real Images. Kaggle. Accessed: December 4, 2024. [Online]. Available: https://www.kaggle.com/datasets/manjilkarki/deepfake-and-real-images?resource=download

[5] T.N. Le, OpenForensics: Multi-Face Forgery Detection And Segmentation In-The-Wild Dataset [V.1.0.0]. (October 13, 2021) . zenodo. https://zenodo.org/records/5528418#.YpdlS2hBzDd

[6] T.N. Le, “OpenForensics: Large-Scale Challenging Dataset For Multi-Face Forgery Detection And Segmentation In-The-Wild” National Institute of Informatics, The Graduate University for Advanced Studies (SOKENDAI), University of Tokyo, July 30 2021, DOI: https://doi.org/10.48550/arXiv.2107.14480

Appendix A:

Model summary of our CNN architecture.

Layer (type)	Output Shape	Param #
random_flip_4 (RandomFlip)	(None, 90, 90, 3)	0
random_flip_5 (RandomFlip)	(None, 90, 90, 3)	0
random_rotation_2 (RandomRotation)	(None, 90, 90, 3)	0
random_zoom_2 (RandomZoom)	(None, 90, 90, 3)	0
random_contrast_2 (RandomContrast)	(None, 90, 90, 3)	0
random_brightness_2 (RandomBrightness)	(None, 90, 90, 3)	0
conv2d_12 (Conv2D)	(None, 90, 90, 32)	896
batch_normalization_14 (BatchNormalization)	(None, 90, 90, 32)	128
conv2d_13 (Conv2D)	(None, 90, 90, 32)	9,248
batch_normalization_15 (BatchNormalization)	(None, 90, 90, 32)	128
max_pooling2d_6 (MaxPooling2D)	(None, 45, 45, 32)	0
dropout_8 (Dropout)	(None, 45, 45, 32)	0
conv2d_14 (Conv2D)	(None, 45, 45, 64)	18,496
batch_normalization_16 (BatchNormalization)	(None, 45, 45, 64)	256
conv2d_15 (Conv2D)	(None, 45, 45, 64)	36,928

batch_normalization_17 (BatchNormalization)	(None, 45, 45, 64)	256
max_pooling2d_7 (MaxPooling2D)	(None, 22, 22, 64)	0
dropout_9 (Dropout)	(None, 22, 22, 64)	0
conv2d_16 (Conv2D)	(None, 22, 22, 128)	73,856
batch_normalization_18 (BatchNormalization)	(None, 22, 22, 128)	512
conv2d_17 (Conv2D)	(None, 22, 22, 128)	147,584
batch_normalization_19 (BatchNormalization)	(None, 22, 22, 128)	512
max_pooling2d_8 (MaxPooling2D)	(None, 11, 11, 128)	0
dropout_10 (Dropout)	(None, 11, 11, 128)	0
flatten_2 (Flatten)	(None, 15488)	0
dense_4 (Dense)	(None, 128)	1,982,592
batch_normalization_20 (BatchNormalization)	(None, 128)	512
dropout_11 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 2)	258

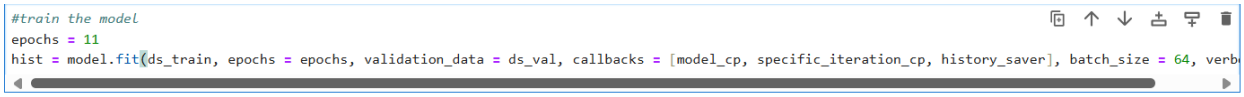
Total params: 6,814,184 (25.99 MB)
Trainable params: 2,271,010 (8.66 MB)
Non-trainable params: 1,152 (4.50 KB)
Optimizer params: 4,542,022 (17.33 MB)

Appendix B:

Code to save training history on json file, best model on a keras file, and every model after every epoch iteration on “90x90Epoch.keras”. The file name for the best model is changed in every new training session to prevent the file from being overwritten by the new training session.

```
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, Callback
es = EarlyStopping(monitor = "val_accuracy", min_delta = 0.01, patience = 10, verbose = 1)
class CustomModelCheckpoint(Callback):
    def __init__(self, filepath, save_freq):
        super(CustomModelCheckpoint, self).__init__()
        self.filepath = filepath
        self.save_freq = save_freq
    def on_epoch_end(self, epoch, logs=None):
        if (epoch + 1) % self.save_freq == 0:  # Save on specific iterations (1-indexed)
            self.model.save(self.filepath.format(epoch=epoch + 1))
#Note: Make a new best Keras file everytime you run the code.
model_cp = ModelCheckpoint(filepath = 'Keras/90x90Best8.keras', monitor = "val_accuracy",
                           save_best_only = True,
                           save_weights_only = False,
                           verbose = 1)
# Define your custom checkpoint for specific iterations
specific_iteration_cp = CustomModelCheckpoint(filepath='Keras/90x90Epoch.keras',
                                              save_freq=1)
import json
import os
import tensorflow as tf  # Ensure you have TensorFlow imported

# Define the custom callback
class HistorySaver(tf.keras.callbacks.Callback):
    def __init__(self, file_path):
        super().__init__()
        self.file_path = file_path
        self.history = []  # Initialize as a list
        self.last_epoch = 0  # Initialize last_epoch

        # Load existing history if the file exists
        if os.path.exists(self.file_path):
            with open(self.file_path, 'r') as file:
                data = json.load(file)
```

```
        # Load existing history if the file exists
        if os.path.exists(self.file_path):
            with open(self.file_path, 'r') as file:
                data = json.load(file)
                self.history = data.get('history', [])
                self.last_epoch = data.get('last_epoch', 0)  # Load last completed epoch

    def on_epoch_end(self, epoch, logs=None):
        # Append the current epoch's logs to the history
        self.history.append({**logs, 'epoch': epoch + 1})  # Store epoch as 1-indexed

        # Save the updated history to the JSON file
        with open(self.file_path, 'w') as file:
             json.dump({'history': self.history, 'last_epoch': epoch + 1}, file, indent=4)

# Usage
file_path = '90x90history.json'
history_saver = HistorySaver(file_path)
# Load the last completed epoch to start from there
start_epoch = history_saver.last_epoch
```



Appendix C:

Code used to create graph 1.

```python
import json
import os
import matplotlib.pyplot as plt
import numpy as np

# Assuming HistorySaver class is already defined as in your code

# Load your history data
file_path = '90x90history.json'
history_saver = HistorySaver(file_path)

# Extract accuracy and validation accuracy from the history saved in history_saver
train_accuracies = [entry['accuracy'] for entry in history_saver.history]
val_accuracies = [entry['val_accuracy'] for entry in history_saver.history]

# Create a range of epochs based on the length of the accuracies
epochs_range = range(1, len(train_accuracies) + 1)

# Plot training and validation accuracy with enhanced styling
plt.figure(figsize=(20, 10))  # Make the plot bigger for poster display
plt.plot(epochs_range, train_accuracies, label='Training Accuracy', marker='o', markersize=8, color='tab:blue', linestyle='-', linewidth=2)
plt.plot(epochs_range, val_accuracies, label='Validation Accuracy', marker='s', markersize=8, color='tab:orange', linestyle='--', linewidth=2)

# Customize the plot
plt.title('Training and Validation Accuracy Over Epochs', fontsize=24, fontweight='bold')  # Bigger title
#plt.suptitle('Training vs. Validation Accuracy during the training process', fontsize=16, fontweight='light')  # Subtitle
plt.xlabel('Epochs', fontsize=18)
plt.ylabel('Accuracy', fontsize=18)

# Improve tick labeling and grid
plt.xticks()  # Optional: Set x-ticks to be every epoch
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.grid(True, which='both', linestyle='--', linewidth=0.5)  # Grid lines
```

```python
# Add a legend with a larger font
plt.legend(fontsize=16)

# Save the plot as a high-quality image
plt.tight_layout()  # Ensure no content is cut off
plt.savefig('training_validation_accuracy.png', dpi=300)  # High resolution for printing

# Show the plot
plt.show()
```

Appendix D:

Using “90x90history.json” to find epoch with highest validation accuracy and validation accuracy at a specific epoch.

```
# Find the highest validation accuracy and its corresponding epoch
highest_val_accuracy = max(val_accuracies)
highest_val_epoch = val_accuracies.index(highest_val_accuracy) + 1  # Adding 1 to match epoch count
# Print the highest validation accuracy and its epoch
print(f'Highest Validation Accuracy: {highest_val_accuracy:.4f} at Epoch {highest_val_epoch}')
```

```
Highest Validation Accuracy: 0.9758 at Epoch 69
```

```
# Specify the epoch you want to check (1-based index)
specific_epoch = 54  # Change this to the desired epoch number

# Get the accuracies for the specific epoch (adjust for 0-based index)
if 1 <= specific_epoch <= len(train_accuracies):
    train_accuracy_at_epoch = train_accuracies[specific_epoch - 1]
    val_accuracy_at_epoch = val_accuracies[specific_epoch - 1]
    print(f'Accuracy at Epoch {specific_epoch}:')
    print(f'Training Accuracy: {train_accuracy_at_epoch:.4f}')
    print(f'Validation Accuracy: {val_accuracy_at_epoch:.4f}')
else:
    print(f'Epoch {specific_epoch} is out of range.')
```

```
Accuracy at Epoch 54:
Training Accuracy: 0.9694
Validation Accuracy: 0.9582
```

Appendix E:

Code used to find which epoch(s) has a specific accuracy on the “90x90history.json” file.

```
# Example: Define the desired validation accuracy threshold
desired_val_accuracy = 0.96877855  # Change this to the validation accuracy you are looking for

# Assume you have a history object from model.fit()
# This contains all the accuracy values during training
#val_accuracies = history.history['val_accuracy']  # Validation accuracy values from training

# Loop through the validation accuracies to find epochs with the desired accuracy
epochs_with_desired_val_accuracy = []
for epoch, val_acc in enumerate(val_accuracies, start=1):  # epoch starts at 1 for human readability
    if val_acc >= desired_val_accuracy and val_acc < 0.969:
        epochs_with_desired_val_accuracy.append(epoch)

# Print the epochs where validation accuracy meets the desired threshold
if epochs_with_desired_val_accuracy:
    print(f"Epochs with validation accuracy >= {desired_val_accuracy}:")
    for epoch in epochs_with_desired_val_accuracy:
        print(f"Epoch {epoch}: Validation Accuracy = {val_accuracies[epoch - 1]:.8f}")
else:
    print(f"No epochs with validation accuracy >= {desired_val_accuracy}")
#Epcoh 47 has best validation and testing performances, 90x90Best3.keras
```

```
Epochs with validation accuracy >= 0.96877855:
Epoch 36: Validation Accuracy = 0.96898144
Epoch 47: Validation Accuracy = 0.96877855
```

Appendix F:

Access to all files mentioned in this report and code used to generate, train, and evaluate for this model is available upon request of Joseph Esteves as listed in cover page.