Real vs. Fake Faces Image Classification

Statistical Deep Learning

2023 Fall

HaeLee Kim

Exploring & Preprocessing the Dataset

1. **Data Source**
- Data Sourced from Kaggle
- Dataset link: https://www.kaggle.com/datasets/hamzaboulahia/hardfakevsrealfaces/data

2. **Data Description**
- Number of real face images : 589
- Number of fake face images : 700
- A total of 1289 images used for the classification

3. **Data Loading and Preprocessing**
- Real and face face images loaded separately
- Assigned Labels: '0' for fake face images, '1' for real face images
- Datasets are combined and shuffled for randomness

4. **Split the Dataset**
- Split ratio: 80% for training and 20% for testing

5. **Image Preprocessing**
- Preprocessed to be consistent in size 300x300 pixels
- Normalized pixel values to the range [0,1]

Convolutional Neural Network Model Generation

```python
num_classes = 2  # fake and real

model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same', input_shape=X_train.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128))  # Reduced to 128 neurons
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

model.summary()
```

```
Model: "sequential"

_____
Layer (type)                Output Shape              Param #
=================================================================
conv2d (Conv2D)             (None, 300, 300, 32)      896

activation (Activation)     (None, 300, 300, 32)      0

max_pooling2d (MaxPooling2   (None, 150, 150, 32)      0
D)

dropout (Dropout)           (None, 150, 150, 32)      0

conv2d_1 (Conv2D)           (None, 150, 150, 64)      18496

activation_1 (Activation)   (None, 150, 150, 64)      0

max_pooling2d_1 (MaxPoolin   (None, 75, 75, 64)        0
g2D)

dropout_1 (Dropout)         (None, 75, 75, 64)        0

flatten (Flatten)           (None, 360000)            0

dense (Dense)               (None, 128)               46080128

activation_2 (Activation)   (None, 128)               0

dropout_2 (Dropout)         (None, 128)               0

dense_1 (Dense)             (None, 2)                 258

activation_3 (Activation)   (None, 2)                 0

=================================================================
Total params: 46099778 (175.86 MB)
Trainable params: 46099778 (175.86 MB)
Non-trainable params: 0 (0.00 Byte)
_____
```

CNN Model Training and Evaluation

```python
y_train = to_categorical(y_train, num_classes=num_classes)
y_test = to_categorical(y_test, num_classes=num_classes)

# Compile the model
opt = Adam(learning_rate=0.0001)
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])

# Data augmentation
datagen = ImageDataGenerator(rotation_range=40, width_shift_range=0.2,
                             height_shift_range=0.2, shear_range=0.2, zoom_range=0.2,
                             horizontal_flip=True, fill_mode='nearest')

# Fit the model
batch_size = 32
epochs = 10
history = model.fit(X_train, y_train, batch_size=batch_size, epochs=epochs,
                    validation_data=(X_test, y_test), verbose=1)
```

```
Epoch 1/10
33/33 [==============================] - 210s 6s/step - loss: 0.6539 - accuracy: 0.8070 - val_loss: 0.2441 - val_accuracy: 0.9845
Epoch 2/10
33/33 [==============================] - 203s 6s/step - loss: 0.1378 - accuracy: 0.9564 - val_loss: 0.2083 - val_accuracy: 0.9457
Epoch 3/10
33/33 [==============================] - 196s 6s/step - loss: 0.1042 - accuracy: 0.9622 - val_loss: 0.1715 - val_accuracy: 0.9767
Epoch 4/10
33/33 [==============================] - 210s 6s/step - loss: 0.1105 - accuracy: 0.9612 - val_loss: 0.1186 - val_accuracy: 0.9922
Epoch 5/10
33/33 [==============================] - 205s 6s/step - loss: 0.1007 - accuracy: 0.9593 - val_loss: 0.1432 - val_accuracy: 0.9922
Epoch 6/10
33/33 [==============================] - 195s 6s/step - loss: 0.0702 - accuracy: 0.9777 - val_loss: 0.0871 - val_accuracy: 0.9922
Epoch 7/10
33/33 [==============================] - 211s 6s/step - loss: 0.0316 - accuracy: 0.9913 - val_loss: 0.0591 - val_accuracy: 0.9922
Epoch 8/10
33/33 [==============================] - 195s 6s/step - loss: 0.0275 - accuracy: 0.9913 - val_loss: 0.0530 - val_accuracy: 0.9961
Epoch 9/10
33/33 [==============================] - 205s 6s/step - loss: 0.0216 - accuracy: 0.9942 - val_loss: 0.0554 - val_accuracy: 0.9961
Epoch 10/10
33/33 [==============================] - 205s 6s/step - loss: 0.0200 - accuracy: 0.9971 - val_loss: 0.0364 - val_accuracy: 0.9961
```

Accuracy & F1 Score of Convolutional Neural Network

```python
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5).astype(int)

y_test = y_test.flatten()
y_pred = y_pred.flatten()

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')

# F1 score
f1 = f1_score(y_test, y_pred, average='binary')
print(f'F1 Score: {f1:.4f}')
```

```
9/9 [==============================] - 11s 1s/step
Accuracy: 0.9961
F1 Score: 0.9961
```

```python
training_acc = history.history['accuracy']
validation_acc = history.history['val_accuracy']
training_loss = history.history['loss']
validation_loss = history.history['val_loss']

epochs = range(1, len(training_acc) + 1)

plt.figure(figsize=(12, 8))

# Plot accuracies
plt.plot(epochs, training_acc, 'bo-', label='Training Accuracy')
plt.plot(epochs, validation_acc, 'ro-', label='Validation Accuracy')
plt.title('Training and Validation Accuracy & Loss')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='center right')

# Plot losses
plt.twinx()  # Create a twin y-axis
plt.plot(epochs, training_loss, 'go-', label='Training Loss')
plt.plot(epochs, validation_loss, 'yo-', label='Validation Loss')
plt.ylabel('Loss')
plt.legend(loc='lower right')

plt.grid()
plt.show()
```
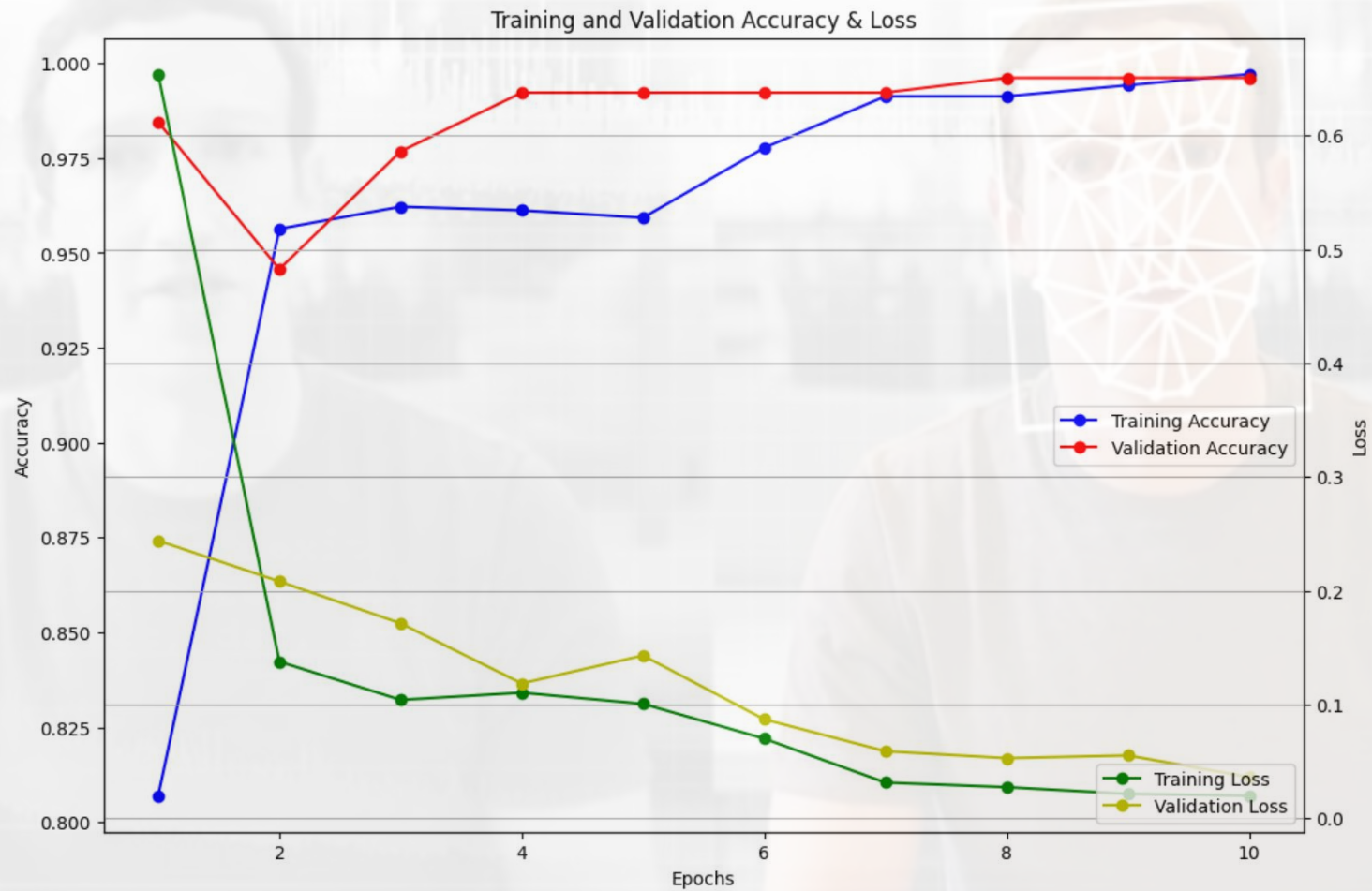
Training and Validation Accuracy & Loss

Conclusion

1. **Training Accuracy**
- Started around 80.7% in the first epoch and steadily improved to nearly 99.7% in the last epoch.
- Indicates effective learning from the training data.

2. **Validation Accuracy**
- Began at 98.4% in the first epoch and stabilized at around 99.2% in later epochs.
- Showed robust generalization to unseen data.

3. **Overall Model Performance**
- Model effectively learnt during training and achieved high accuracy in both training and validation datasets.

4. **Limitation**
- Model's impressive accuracy and generalization could be attributed to the limited dataset.
- A larger and more diverse dataset might provide a better understanding of the model's performance.

5. **Github**
- Code link: https://github.com/HL-Kim/Deep-Learning
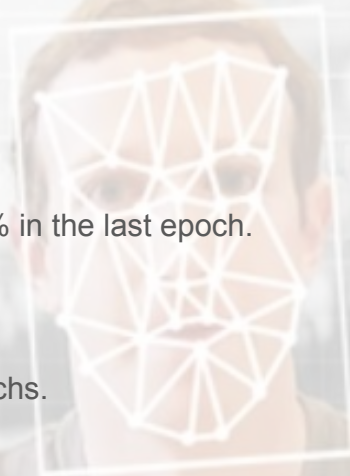
Thank you to
all the real, fake, and even..
deep fake faces for joining today.

Remember.
Whether you are real or not,
Your presence is appreciated…