Toenail Infection Detector

Final Project Report

ECE-GY 6143 Machine Learning

Abdul Samad Zaheer Khan
ak9943@nyu.edu

1. QUAD CHART

Problem Detecting if a toenail is infected or not using the image of the toenail.	**Findings** We can solve this problem using binary classification machine learning models to detect if a toenail is infected or benign. This model can be further trained to classify the infection, but will require more training resources.
Design We are using Convolutional Neural Networks(CNNs) to train a model to detect the infection. All data was classified by me by downloading all the images from Google and other websites as there was no existing dataset available.	**Conclusion** The accuracy of the model generated can be improved by using more data and other resources. Another possible improvement is that we can further train the model to classify the type of infection based on the image.

1.1 Motivation

Many people have fungal infection on their toe but it develops so slowly that most people don't identify the infection and carry on. These infections are very difficult to get rid of and require lots of medications to cure. Hence we can prevent the infection by just using the model to detect the infection and the current stage of the infection. This is a cheap precautionary measure which can be used by individuals to stay safe.

1.2 Why solve this problem using Machine Learning Models?

Machine learning models once trained can be distributed easily with minor tweaks required over time to become more accurate. No other method can be used to detect infections in such a affordable way, as other ways are much more expensive such as professional doctor. We have to keep in mind that this is just a preventive measure and if an infection is detected then taking professional help is necessary.

1.3 Machine Learning Models Used

We are using Convolutional Neural Networks(CNNs) because we are using images to train the model. CNNs are known to find reliable patterns in the images to predict. Also training of images does not take a lot of time as there are very few images.

1.4 Summary

The model predicts accurately in most of the cases but can be improved using more training data and more processing power to extract excellent patterns. The model is vary basic and only detects if a person is infected or not. The data can be trained to predict the type of infection which can be more useful to individuals using the model to check for infection.

2. CONTRIBUTIONS

2.1 Dataset selection

Since there was no dataset available online I created a new dataset with approximately 250 images which I downloaded from Google and other websites. Then I classified the data individually if it is infected or benign. I will upload the dataset along with my code to my Github repository(https://github.com/abdulllkhan/toenail_infection) to verify the dataset.

2.2 Dataset Preprocessing

Since it is not a standardized dataset I had to resize all the images to 64*64 resolution and also converted the image from RGB to Grayscale for faster model generation.

2.3 Implementation of filtering techniques

All images that had a lower resolution than 64*64 are discarded to get proper results.

2.4 Training and Testing Procedures

Trained and tested data were divided in 80-20 ratio. The dataset were randomly distributed to fade any unnecessary numbering patterns.

2.5 Performance Metrics

Inbuilt accuracy metrics were used to evaluate the accuracy of the model and further improve to get the best model

2.6 Optimizers Used

Used Adam for CNNs model which is available in the K**eras** library and doesn't require any exclusive import.

2.7 Epoch Configuration Used

20 epochs were used for uniform training of the model to get the best model.

2.8 Visualization

Matplotlib library was used to plot the graph for accuracy and loss for the training and testing data. All 20 iterations of the model are plotted by storing the results in the history variable.

3. PROJECT REPORT

3.1 Problem Statement

Fungal infections on the toes often progress slowly, making it challenging for individuals to recognize the infection in its early stages. Many people have fungal infection on their toe but it develops so slowly that most people don't identify the infection and carry on. These infections are very difficult to get rid of and require lots of medications to cure. Hence we can prevent the infection by just using the model to detect the infection and the current stage of the infection. This is a cheap precautionary measure which can be used by individuals to stay safe.

3.2 Data Exploration

Since there was no dataset available online I created a new dataset with around 250 images which I downloaded from Google and other websites. Then I classified the data individually if it is infected or benign. I will upload the dataset along with my code to my Github repository(https://github.com/abdulllkhan/toenail_infection) to verify the dataset.

No features are defined for the images. Infected toenail images and healthy toenail images are present in a different folders. The images are assigned prediction values in the code block. Healthy toenail images are given the value 0 and infected toenail images are given the value 0. Each image is 64*64 after resizing. Also the images are converted from RGB to Grayscale for easier assessment.

3.3 Project Overview

3.3.1 Objective

The goal of this project is to predict whether a toenail is infected or not with enough confidence that it is acceptable. We are using CNNs which work best for image classification and the images used are correctly classified to detect the infection. The feature extraction method used by CNNs is very accurate and we can use the accuracy values to improve the model accuracy over multiple iterations.

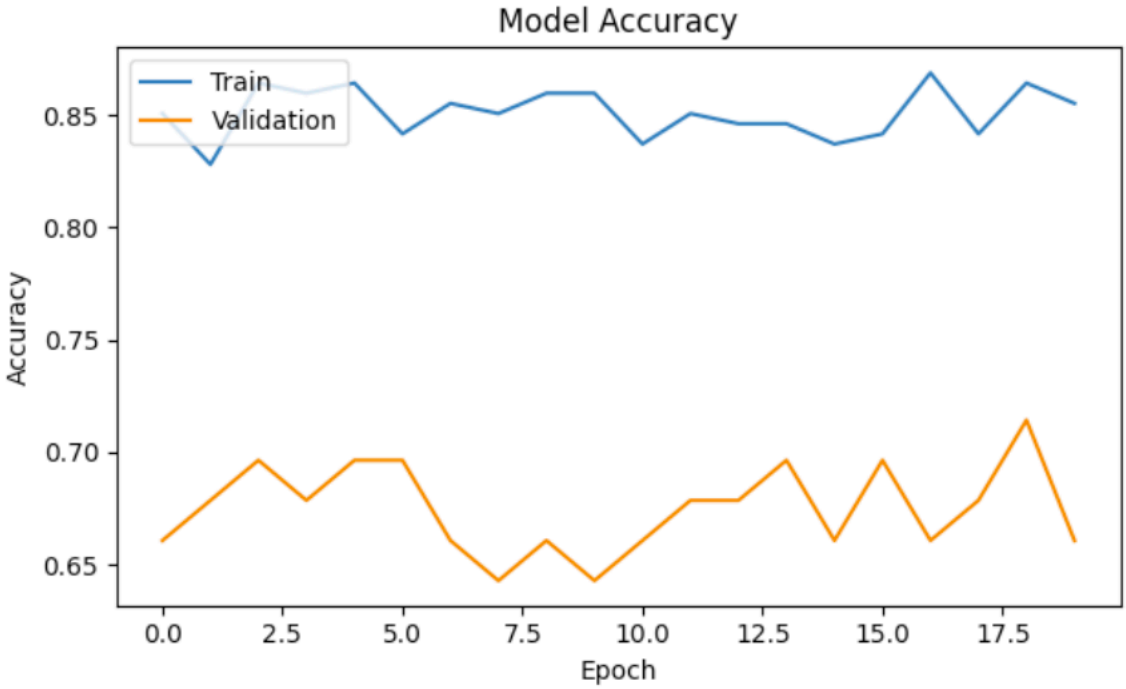
3.3.2 Evaluation and Comparison

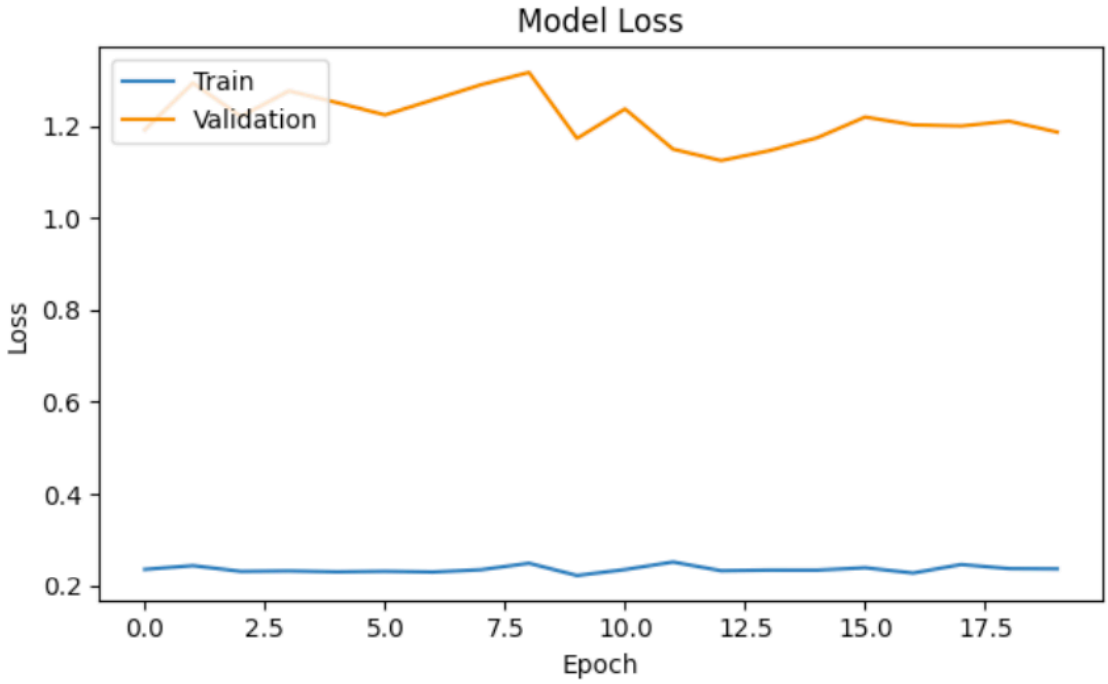
The model generates accuracy metrics in every iteration of the model fitting. This can be used to improve on the features for better fitting for the next model. It is run for 20 iterations to get the best result possible. The best model is used to predict the values of any future inputs.

3.3.3 Results and Visualization

The generated model can predict if a person is infected or not with approximately 70% confidence. Even though it is not acceptable this can be improved if more data was available to train the model.

The data cannot be visualized as all the data is images. But the accuracy and loss of each model is visualized for understanding hoe the final model was selected.


Model Accuracy
Train
Validation
Accuracy
0.85
0.80
0.75
0.70
0.65
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch


Model Loss
Train
Validation
Loss
1.2
1.0
0.8
0.6
0.4
0.2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch

3.4 Methods Used to Code

NumPy was used to store the images in array form where every pixel was represented as a bit. Pandas could have been used to get the image files but I opted for basic fetching methods using os library. Convolutional Neural Networks(CNNs) from the Keras library are used to fit the data and generate the optimal model over 20 epochs. The inbuilt accuracy value are used to decide the best model to use for for the prediction of real-world data.

3.5 Analysis

This code sets up a binary classification model for detecting toenail infections using a CNN. It demonstrates the complete process, from data loading to model training and evaluation. The early stopping and training history plotting contribute to effective model development and analysis.

3.5.1 Data Loading and Preprocessing

Images are loaded from the 'infected' and 'benign' folders and stored in separate lists. Images and labels are combined and converted to NumPy arrays. Data is shuffled, and then split into training and testing sets. Images are reshaped for compatibility with the CNN model.

3.5.2 Model Definition

A simple CNN model is defined using Keras. The model consists of convolutional and pooling layers followed by fully connected layers. Binary cross-entropy is used as the loss function, and accuracy is chosen as the metric.

3.5.3 Model Training

The model is trained using the training set. Early stopping is implemented to monitor validation accuracy and prevent overfitting. The training history is stored for later analysis.

3.5.4 Training History Plotting

The training and validation accuracy and loss are plotted over epochs to visualize model performance.

3.6 Conclusion

The accuracy of the not acceptable in real world scenarios but with more data and better feature extraction the model can be prepared to be acceptable for real-world applications.

In conclusion, while the provided code lays the foundation for toenail infection classification, there are opportunities for refinement and enhancement. Future iterations could involve addressing dataset limitations, exploring advanced architectures, and incorporating a more comprehensive evaluation strategy.

3.7 References

1. google.com for Keras and other model related queries.
2. stackoverlfow.com to resolve errors faced during model training.
3. Previous homework and lecture notes.

The code and dataset is available in my personal Github: https://github.com/abdulllkhan/toenail_infection

4. SOURCE CODE

Code was written from scratch and all functions were implemented by me. Any code that had issues and was resolved using google and stack overflow is highlighted.

The code and dataset is available in my personal Github: https://github.com/abdulllkhan/toenail_infection

infection-classifer

December 24, 2023

[75]:
```
# Abdul Samad Zaheer Khan, ak9943@nyu.edu


# importing all the necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

[77]:
```
import os
from skimage.io import imread

# to store infected toe-nail images
infected_images = []
# fetching from the folder
for img in os.listdir('data/infected'):
    infected_images.append(imread('data/infected/' + img))

# to store healthy toe-nail images
benign_images = []
# fetching from the folder
for img in os.listdir('data/benign'):
   benign_images.append(imread('data/benign/' + img))

X = infected_images + benign_images   # storingg the data

# assigning labels to seperate images to classify infected and benign toe-nail␣
 ↪images
y = [1]*len(infected_images) + [0]*len(benign_images)
```

[131]:
```
# preprocessing the images

from skimage.io import imread
from skimage.transform import resize
from skimage.color import rgb2gray
import os

# fucntion to pre process the images
# resizing the images to 64*64 dimensions otherwise every image should be␣
 ↪trained individually
def preprocess_image(img_path):
    img = imread(img_path)

    # removing alpha channel if it exists
    if img.shape[-1] == 4:
        img = img[:, :, :3]

    # resizing
    img_resized = resize(img, (64, 64), anti_aliasing=True)

    # converting the image to grayscale
    if len(img_resized.shape) == 3:
        img_resized = rgb2gray(img_resized)

    return img_resized

# preprocess infected toe-nail images
infected_images = [preprocess_image(os.path.join('data', 'infected', img)) for␣
 ↪img in os.listdir(os.path.join('data', 'infected'))]

# preprocess healthy toe-nail images
benign_images = [preprocess_image(os.path.join('data', 'benign', img)) for img␣
 ↪in os.listdir(os.path.join('data', 'benign'))]

# checking if all images have the same shape
image_shape = infected_images[0].shape
infected_images = [img if img.shape == image_shape else resize(img,␣
 ↪image_shape, anti_aliasing=True) for img in infected_images]
benign_images = [img if img.shape == image_shape else resize(img, image_shape,␣
 ↪anti_aliasing=True) for img in benign_images]

# combinign the pre-processsed images
X = infected_images + benign_images

# converting to NumPY arrays
X = np.array(X)
y = np.array([1] * len(infected_images) + [0] * len(benign_images))
```

[132]:
```python
# printing the labels and the number of labels
print(y)
print("Total number of images are:", len(y))

print("\n")
# printing total number of infected and bening toe-nail images
print("Total number of infected toe-nail images are:", len(infected_images))
print("Total number of healthy toe-nail images are:", len(benign_images))
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
Total number of images are: 277


Total number of infected toe-nail images are: 191
Total number of healthy toe-nail images are: 86
```

[133]:
```python
# splitting the data into test train components

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 42)
```

[134]:
```python
print(len(X_train))
print(len(y_train))
print(y_train)
```

```
221
221
[0 1 1 1 0 0 1 1 1 0 0 1 1 1 1 1 0 1 1 0 0 1 1 1 1 1 0 0 1 0 1 1 0 1 1 1 0
 1 0 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 0 1 0 1 1 1 1 1 1
 1 1 1 1 1 1 0 1 1 1 0 1 1 0 1 1 1 0 1 0 1 0 1 0 0 1 1 1 1 0 1 1 0 0 1 1 1
 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 1 1 1 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 1 1
 0 1 1 1 1 1 0 1 1 1 1 0 1 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0
 1 1 1 1 0 1 0 1 1 0 1 0 1 1 0 1 0 0 1 1 0 0 1 1 1 1 1 0 1 0 1 1 1 1 0 1]
```

[135]:
```python
# imprting the CNN model because we are using images

from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
```

[136]:
```
# adding features to the model architecture

# calculating the necessary arguments
img_height, img_width = X_train[0].shape[:2]
input_shape = (64, 64, 1)

# defining model architecture
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

[137]:
```
# compiling the model

model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
```

[138]:
```
# model summary
model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 62, 62, 32)        320

 max_pooling2d_6 (MaxPoolin  (None, 31, 31, 32)        0
 g2D)

 flatten_6 (Flatten)         (None, 30752)             0

 dense_8 (Dense)             (None, 64)                1968192

 dense_9 (Dense)             (None, 1)                 65

=================================================================
Total params: 1968577 (7.51 MB)
Trainable params: 1968577 (7.51 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[156]:
```
# fitting the model on the traning dataset

# converting data from list datatype of Numpy array
X_train = np.array(X_train)
```

```
y_train = np.array(y_train)

history = model.fit(X_train, y_train, epochs = 20, validation_data = (X_test,␣
 ↪y_test))
```

```
Epoch 1/20
7/7 [==============================] - 0s 14ms/step - loss: 0.2360 - accuracy:
0.8507 - val_loss: 1.1918 - val_accuracy: 0.6607
Epoch 2/20
7/7 [==============================] - 0s 13ms/step - loss: 0.2441 - accuracy:
0.8281 - val_loss: 1.2933 - val_accuracy: 0.6786
Epoch 3/20
7/7 [==============================] - 0s 12ms/step - loss: 0.2317 - accuracy:
0.8643 - val_loss: 1.2206 - val_accuracy: 0.6964
Epoch 4/20
7/7 [==============================] - 0s 12ms/step - loss: 0.2327 - accuracy:
0.8597 - val_loss: 1.2761 - val_accuracy: 0.6786
Epoch 5/20
7/7 [==============================] - 0s 13ms/step - loss: 0.2307 - accuracy:
0.8643 - val_loss: 1.2507 - val_accuracy: 0.6964
Epoch 6/20
7/7 [==============================] - 0s 13ms/step - loss: 0.2318 - accuracy:
0.8416 - val_loss: 1.2239 - val_accuracy: 0.6964
Epoch 7/20
7/7 [==============================] - 0s 12ms/step - loss: 0.2303 - accuracy:
0.8552 - val_loss: 1.2566 - val_accuracy: 0.6607
Epoch 8/20
7/7 [==============================] - 0s 12ms/step - loss: 0.2353 - accuracy:
0.8507 - val_loss: 1.2896 - val_accuracy: 0.6429
Epoch 9/20
7/7 [==============================] - 0s 13ms/step - loss: 0.2493 - accuracy:
0.8597 - val_loss: 1.3164 - val_accuracy: 0.6607
Epoch 10/20
7/7 [==============================] - 0s 13ms/step - loss: 0.2227 - accuracy:
0.8597 - val_loss: 1.1728 - val_accuracy: 0.6429
Epoch 11/20
7/7 [==============================] - 0s 13ms/step - loss: 0.2357 - accuracy:
0.8371 - val_loss: 1.2370 - val_accuracy: 0.6607
Epoch 12/20
7/7 [==============================] - 0s 12ms/step - loss: 0.2520 - accuracy:
0.8507 - val_loss: 1.1495 - val_accuracy: 0.6786
Epoch 13/20
7/7 [==============================] - 0s 12ms/step - loss: 0.2331 - accuracy:
0.8462 - val_loss: 1.1247 - val_accuracy: 0.6786
Epoch 14/20
7/7 [==============================] - 0s 12ms/step - loss: 0.2345 - accuracy:
0.8462 - val_loss: 1.1460 - val_accuracy: 0.6964
```

```
Epoch 15/20
7/7 [==============================] - 0s 12ms/step - loss: 0.2343 - accuracy:
0.8371 - val_loss: 1.1742 - val_accuracy: 0.6607
Epoch 16/20
7/7 [==============================] - 0s 12ms/step - loss: 0.2396 - accuracy:
0.8416 - val_loss: 1.2193 - val_accuracy: 0.6964
Epoch 17/20
7/7 [==============================] - 0s 12ms/step - loss: 0.2281 - accuracy:
0.8688 - val_loss: 1.2024 - val_accuracy: 0.6607
Epoch 18/20
7/7 [==============================] - 0s 14ms/step - loss: 0.2466 - accuracy:
0.8416 - val_loss: 1.1999 - val_accuracy: 0.6786
Epoch 19/20
7/7 [==============================] - 0s 13ms/step - loss: 0.2377 - accuracy:
0.8643 - val_loss: 1.2103 - val_accuracy: 0.7143
Epoch 20/20
7/7 [==============================] - 0s 12ms/step - loss: 0.2371 - accuracy:
0.8552 - val_loss: 1.1866 - val_accuracy: 0.6607
```

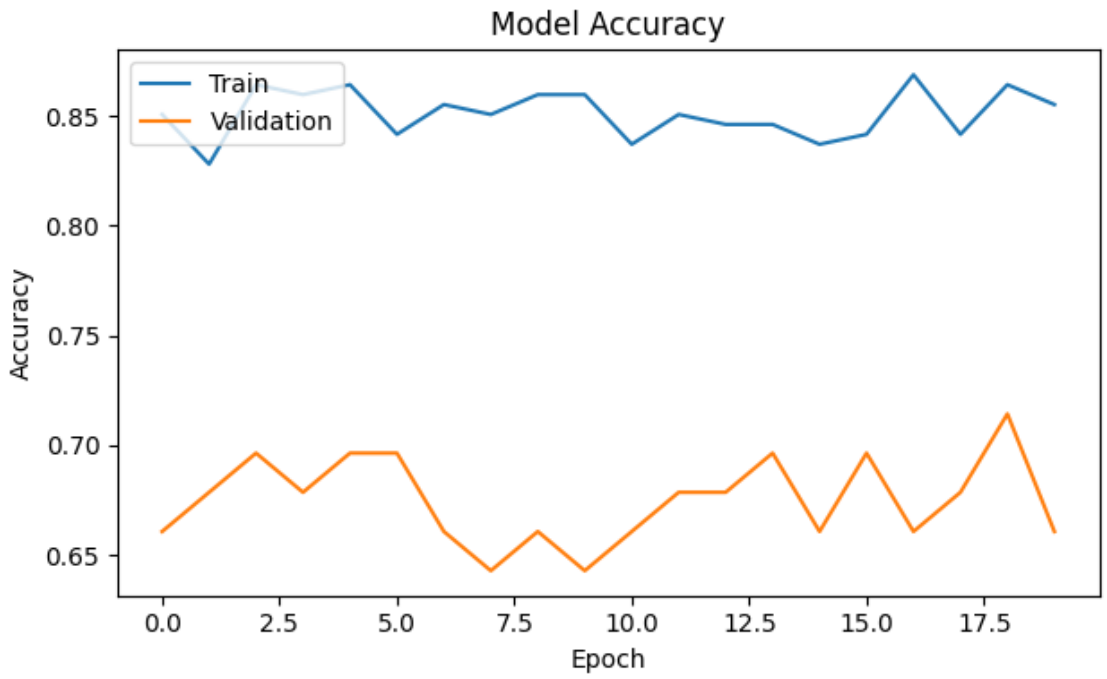
[157]:
```python
import matplotlib.pyplot as plt

# plotting model acccuracy

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')


plt.tight_layout()
plt.show()
```

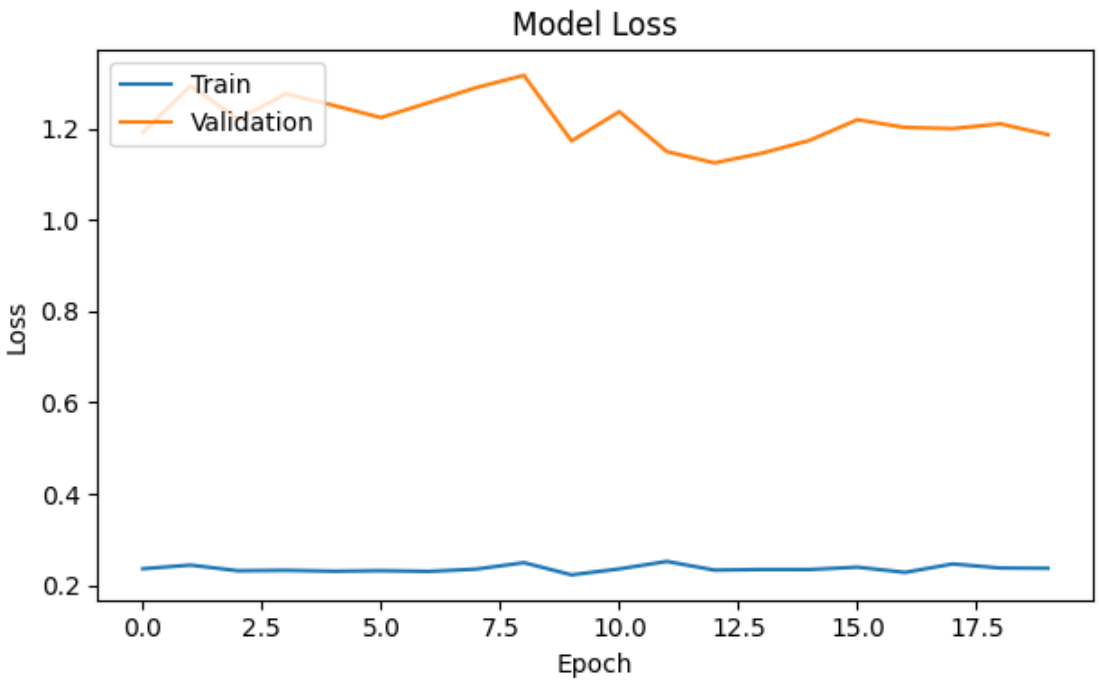


[158]:
```
# plotting model loss

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.tight_layout()
plt.show()
```



[159]:
```
final_training_accuracy = history.history['accuracy'][-1]
print(f"Final Training Accuracy:", final_training_accuracy)

# Get the final validation accuracy
final_testing_accuracy = history.history['val_accuracy'][-1]
print("Final Testing Accuracy:", final_testing_accuracy)
```

```
Final Training Accuracy: 0.8552036285400391
Final Testing Accuracy: 0.6607142686843872
```

[160]:
```
print("done")
```

```
done
```

[ ]:
```

```