Image Classification using Convolutional Neural Networks

Jeeven Navaah

Abstract

There are many different approaches used in machine learning to classify images. In this project, two different methods were chosen to classify a dataset containing 800 images. The first method used to classify the images was logistic regression. Following this, a convolutional neural network was used to classify the same dataset. The goal was to use a basic model to establish a baseline in order to determine if there were any gains in performance when using a more sophisticated model to classify the images. The results showed significant improvements in overall performance by using a convolutional neural network over logistic regression.

Introduction

The purpose of this project was to explore the use of machine learning in image classification. The goal was to solve a binary classification problem to determine whether a photograph was taken outdoors in daylight. Two different models were used in this project. At first logistic regression was used to classify the images. This was a naive approach that was used to establish a baseline performance metric. The second attempt at classifying the images made use of a more sophisticated model.

There have been many machine learning techniques used for classification, with models such as support vector machines being considered state-of-the-art for many years. However, in the past decade the use of deep neural networks, often referred to as deep learning, has gained widespread adoption and popularity. A major turning point for neural networks came in 2012 when AlexNet won the ImageNet image classification competition by a large margin over competitors using more traditional methods. This major breakthrough was due to the fact that AlexNet made use of a convolutional neural network to classify the images [1]. A convolutional neural network is a type of neural network pioneered by Yann LeCun mainly for use in image classification tasks. Convolutional neural networks found early success in classifying handwritten digits with high accuracy [2]. This makes convolution neural networks ideal for classifying the images in this project and was therefore chosen as the second model.

Data

The images used in this project came from the “Personal Columbia” dataset [3]. This dataset contains 800 colour images taken by professional cameras in various locations around New York and Boston. The dataset includes a diverse collection of images taken both indoors and outdoors, in many different lighting conditions. Each image is a jpeg file containing three colour channels, with pixel values ranging from 0 to 255. The images come with an accompanying CSV file with metadata including the classification label of each image. For this project, the primary concern was the classification of outdoor images taken during the day, which were labeled “outdoor-day”. For logistic regression, the classification labels were extracted from the CSV file directly and all the images were contained in one folder. However, the TensorFlow library used for convolutional neural networks required the images belonging to different classes to be put into their own folder. For this purpose, there was a second folder where the images classified as “outdoor-day” were contained in its own sub-folder and the remaining images contained in another sub-folder.

Methodology

To perform logistic regression, the labels of the 800 images were extracted from the CSV file and initialized in a response vector of length 800. The images labeled “outdoor-day” were assigned the value 1 and the rest were assigned the value 0. For the image data, the median values of all the pixels in each of the three channels were calculated. These three values were used as features in constructing the design matrix, with a dimension of 800x3. However, before running logistic regression, the dataset was partitioned using an 80/20 split. As a result, 640 images were put in the training set and the remaining 160 images were put in the testing set. The training set was used to perform logistic regression and the testing set was used to evaluate the model.

Building a convolutional neural network requires many decisions to be made concerning the architecture and hyperparameters. Unfortunately, this is not an exact science and many of these decisions have to be made through experimentation, requiring a great deal of time and resources. Although there are a lot of deep and complex convolutional neural networks in use today, to keep things simple, the architecture used in this project was based on LeNet-5 [4]. As shown in Figure 1, the first layer of LeNet-5 is a convolutional layer followed by a pooling layer. At layer three is another convolutional layer which is once again followed by a pooling layer. These layers connect to two fully connected layers and then an output layer at the end. However, a few modifications to the LeNet-5 architecture were made for this project. LeNet-5 was designed to take a 32x32 pixel image as input. The images used in this project were 256x256 pixels with three colour channels (256x256x3). This required increasing the input size of the convolutional neural network and

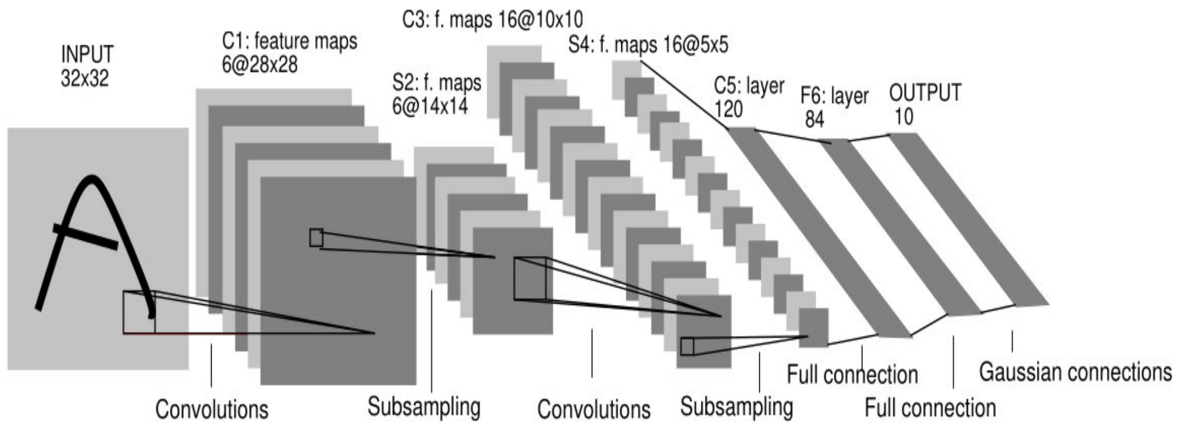


Figure 1: Architecture of LeNet-5 (LeCun et al., 1998).

adjusting the size of the following layers to compensate. Since LeNet-5 was built to recognize the digits from 0 to 9, it had a final softmax layer with 10 possible outputs. This was replaced with a sigmoid layer for the purpose of binary classification, which is what was required for this project.

Since the development of LeNet-5 in 1995, there have been many advancements that have led to faster and better-performing neural networks. Fortunately, it was possible to integrate many of these advancements into the convolutional neural network used in this project, without losing the simplicity of the original model. LeNet-5 originally made use of the tanh function as the activation function. This was replaced by the ReLU activation function for this project. The ReLU activation function has been shown to outperform other activation functions and is widely used in neural networks today [5]. Also, there were additional dropout layers added to the convolutional neural network. Dropout is a way to add regularization to the neural network, which reduces overfitting and improves generalization [6]. Although regularization techniques such as dropout are most beneficial for complex neural networks, there is not much of a drawback to using them in simpler neural networks. The final modification made to the convolutional neural network was the use of the Adam optimizer for training. Adam is an algorithm that improves upon stochastic gradient descent by dynamically adjusting the learning rate to significantly speed up training [7].

In the preprocessing step, all the images in the dataset were normalized and converted to 256x256 pixels in order to fit the input layer of the convolutional neural network. The images were also randomly assigned to a training set and testing set, similar to the dataset used in the logistic regression. However, this time an additional validation set containing 96 images (12% of the dataset) was partitioned from the training set. The validation set was created to allow the TensorFlow library to better tune the hyperparameters during the training phase. The training set and validation

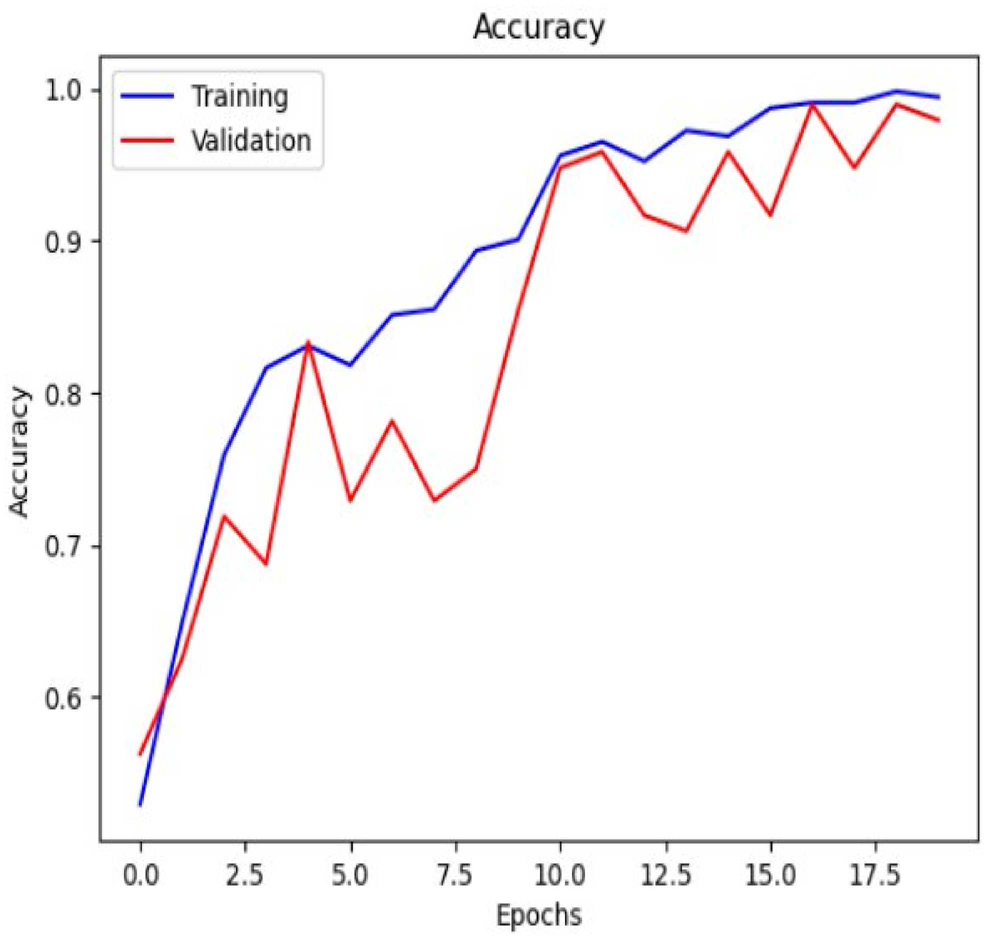


Figure 2: Accuracy of the training set and validation set at each epoch during the training of the convolutional neural network.

set were then used to train the convolutional neural network for a total of 20 epochs. The training progress of the convolutional neural network is shown in Figure 2, which compares the accuracy achieved by the model on both the training set and validation set at each epoch. Finally, the model was evaluated on the testing set.

Results

For this project sensitivity, specificity and accuracy were the metrics used to evaluate and compare the performance of the two models. The logistic regression model had a sensitivity of 45.61%, a specificity of 89.32% and an accuracy of 73.75%. The convolutional neural network had a sensitivity of 98.31%, a specificity of 99.01% and an accuracy of 98.75%.

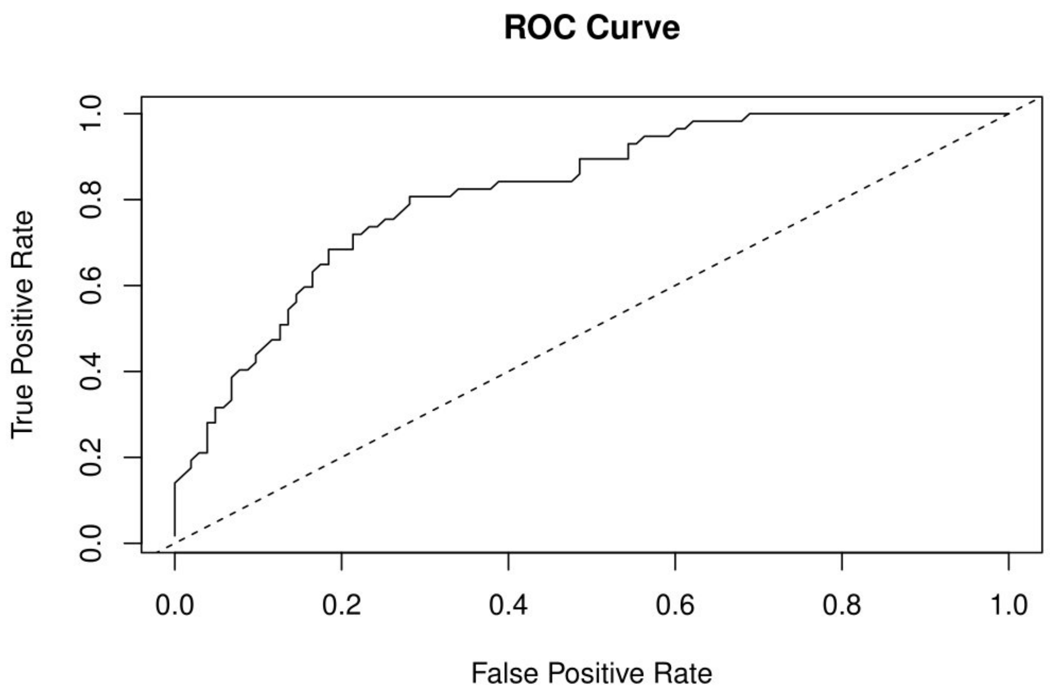


Figure 3: ROC curve for the logistic regression model.

Analysis

The convolutional neural network outperformed the logistic regression model in every metric measured. The logistic regression model had a very low score on sensitivity, which meant the model was producing many false negatives by failing to recognize the "outdoor-day" images. The low sensitivity could have been corrected to an extent by tuning the prediction threshold, which was set at 0.5 for this project. However, based on the ROC curve in Figure 3, it is clear that no amount of tuning would improve the overall performance of the model. A major reason for this was that the logistic regression model made use of only three features, which were the median values of the pixels in each of the three colour channels. Although it is possible to predict daylight from the intensity of these median values, there was a lot of information contained in the pixels that was being thrown away.

On the other hand, the convolutional neural network was able to take all the pixels in the image as input while keeping the size of the model manageable. The numerous filters in the convolutional layers of the network were able to extract many features from the images, which aided the network in making its final predictions. An additional advantage of the convolutional neural network was that it was not necessary to determine which features needed to be extracted from the image ahead of time. This was contrary to the logistic regression model, where a deliberate decision was made to extract the median values of the pixels. In the convolutional neural network, the training phase allowed the network to automatically choose the most optimum features necessary to make the best predictions. This explains how the convolutional neural network was able to achieve near-perfect accuracy in the image classification task.

Conclusion

The results obtained in this project suggest that there are significant gains in performance that can be attained by using convolutional neural networks for image classification tasks. It is also important to note that the convolution neural network used in this project was able to achieve great results despite being severely constrained in terms of its complexity and training time. In practice, if enough resources were available, it would be possible to achieve even better performance with deeper convolutional neural networks and longer training time.

References

[1] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems 25 (2012).

[2] LeCun, Yann, et al. "Handwritten digit recognition with a back-propagation network." Advances in neural information processing systems 2 (1989).

[3] Ng, T. T., et al. "Columbia photographic images and photorealistic computer graphics dataset ADVENT." Columbia University, New York, NY, USA (2004).

[4] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

[5] Schmidhuber, Jürgen. "Deep learning in neural networks: An overview." Neural networks 61 (2015): 85-117.

[6] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." The journal of machine learning research 15.1 (2014): 1929-1958.

[7] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

Image Classification using Logistic Regression

Loading the Dataset

```
pm <- read.csv("photoMetaData.csv")
n <- nrow(pm)

y <- as.numeric(pm$category == "outdoor-day")
X <- matrix(NA, ncol=3, nrow=n)

for(j in 1:n) {
  img <- readJPEG(paste0("./columbiaImages/", pm$name[j]))
  X[j, ] <- apply(img, 3, median)
}
```

Partitioning the Dataset

```
training <- sample(1:800, 640)

X_train <- X[training,]
y_train <- y[training]

X_test <- X[-training,]
y_test <- y[-training]
```

Performing Logistic Regression on the Training Set

```
model <- glm(y_train ~ X_train, family=binomial)
summary(model)
```

```
##
## Call:
## glm(formula = y_train ~ X_train, family = binomial)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -2.0634     0.2505  -8.236  < 2e-16 ***
## X_train1     -6.9180     1.6602  -4.167 3.09e-05 ***
## X_train2      2.8237     2.6081   1.083    0.279
## X_train3      9.9514     1.8068   5.508 3.64e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 823.67  on 639  degrees of freedom
## Residual deviance: 640.80  on 636  degrees of freedom
## AIC: 648.8
##
## Number of Fisher Scoring iterations: 5
```

Sensitivity, Specificity and Accuracy

```r
predictions <- 1 / (1 + exp(-1 * cbind(1,X_test) %*% coef(model)))
predictions_class <- as.numeric(predictions >= 0.5)

true_positive <- sum(predictions_class == 1 & y_test == 1)
true_negative <- sum(predictions_class == 0 & y_test == 0)

sensitivity <- true_positive / sum(y_test == 1)
specificity <- true_negative / sum(y_test == 0)
accuracy <- (true_positive + true_negative) / length(y_test)

sprintf("Sensitivity: %s, Specificity: %s, Accuracy: %s", sensitivity, specificity, accuracy)
```

```
## [1] "Sensitivity: 0.456140350877193, Specificity: 0.893203883495146, Accuracy: 0.7375"
```

ROC Curve

```r
roc <- function(y, pred) {
  alpha <- quantile(pred, seq(0, 1, by=0.01))
  N <- length(alpha)

  sens <- rep(NA,N)
  spec <- rep(NA,N)

  for(i in 1:N) {
    pred_class <- as.numeric(pred >= alpha[i])
    sens[i] <- sum(pred_class == 1 & y == 1) / sum(y == 1)
    spec[i] <- sum(pred_class == 0 & y == 0) / sum(y == 0)
  }

  return(list(fpr=1-spec, tpr=sens))
}

roc <- roc(y_test, predictions)

plot(roc$fpr,
     roc$tpr,
     main="ROC Curve",
     xlab="False Positive Rate",
     ylab="True Positive Rate",
     type="l")

abline(0, 1, lty="dashed")
```

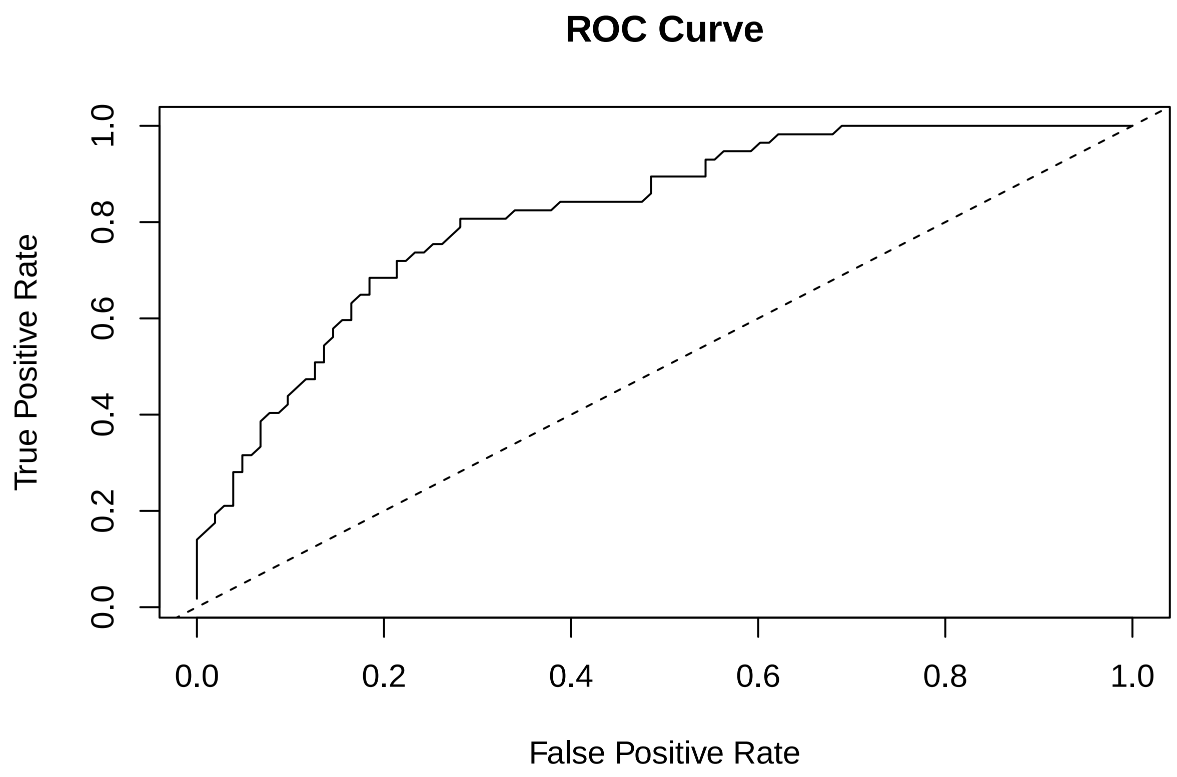

ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

Image Classification using Convolutional Neural Networks

In [1]:
```
import tensorflow as tf
import numpy as np
from matplotlib import pyplot as plt
```

Importing the Dataset

In [2]:
```
data = tf.keras.utils.image_dataset_from_directory('columbiaImages-cnn', batch_size=32, image_size=(256, 256), shuffle=True)
```

```
Found 800 files belonging to 2 classes.
```

Normalizing the Dataset

In [3]:
```
data = data.map(lambda x, y: (x / 255, y))
```

In [4]:
```
# check the range of the values
batch = data.as_numpy_iterator().next()
print(f'Minimum Value: {batch[0].min()}, Maximum Value: {batch[0].max()}')
```

```
Minimum Value: 0.0, Maximum Value: 1.0
```

Partitioning the Dataset

In [5]:
```
# partition data by batches of 32 images
num_batches = len(data)
training_size = int(num_batches * 0.68)
validation_size = int(num_batches * 0.12)
testing_size = int(num_batches * 0.20)

training_set = data.take(training_size)
validation_set = data.skip(training_size).take(validation_size)
testing_set = data.skip(training_size + validation_size).take(testing_size)
```

Building the Convolutional Neural Network

In [6]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
```

In [7]:
```
model = Sequential()

model.add(Conv2D(16, (3,3), 1, activation='relu', input_shape=(256,256,3)))
model.add(MaxPooling2D())
model.add(Dropout(0.25))

model.add(Conv2D(32, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())
model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(256, activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))

model.add(Dense(1, activation='sigmoid'))
```

In [8]:
```
model.compile(optimizer='adam', loss=tf.keras.losses.BinaryCrossentropy(), metrics=['accuracy'])
```

In [9]:
```
model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 254, 254, 16)	448
max_pooling2d (MaxPooling2D)	(None, 127, 127, 16)	0
dropout (Dropout)	(None, 127, 127, 16)	0
conv2d_1 (Conv2D)	(None, 125, 125, 32)	4,640
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 32)	0
dropout_1 (Dropout)	(None, 62, 62, 32)	0
flatten (Flatten)	(None, 123008)	0
dense (Dense)	(None, 256)	31,490,304
dropout_2 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32,896
dropout_3 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 1)	129

```
Total params: 31,528,417 (120.27 MB)
Trainable params: 31,528,417 (120.27 MB)
Non-trainable params: 0 (0.00 B)
```

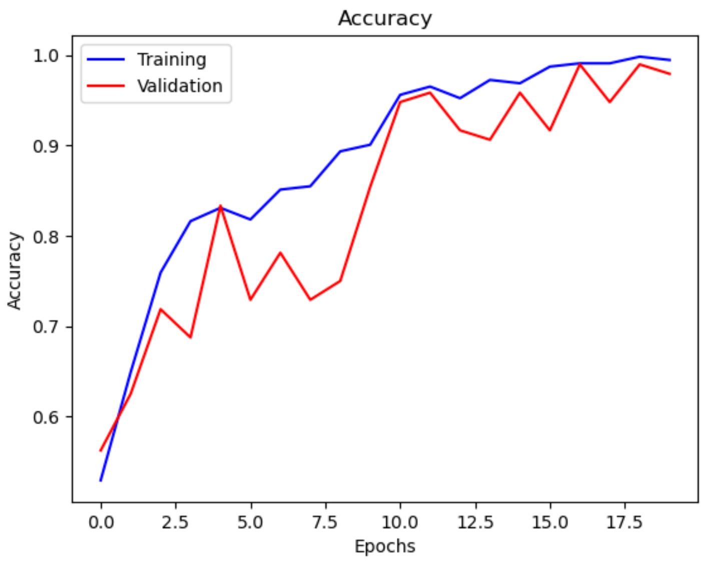
Training the Model

```
In [10]: history = model.fit(training_set, epochs=20, validation_data=validation_set)
```

```
Epoch 1/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 5s 246ms/step - accuracy: 0.5265 - loss: 3.6998 - val_accuracy: 0.5625 - val_loss: 0.7325
Epoch 2/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 239ms/step - accuracy: 0.6233 - loss: 0.7076 - val_accuracy: 0.6250 - val_loss: 0.5223
Epoch 3/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 239ms/step - accuracy: 0.7547 - loss: 0.5162 - val_accuracy: 0.7188 - val_loss: 0.4660
Epoch 4/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 239ms/step - accuracy: 0.8235 - loss: 0.4220 - val_accuracy: 0.6875 - val_loss: 0.5271
Epoch 5/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 237ms/step - accuracy: 0.8320 - loss: 0.3989 - val_accuracy: 0.8333 - val_loss: 0.4132
Epoch 6/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 237ms/step - accuracy: 0.7887 - loss: 0.4567 - val_accuracy: 0.7292 - val_loss: 0.4244
Epoch 7/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 238ms/step - accuracy: 0.8259 - loss: 0.3684 - val_accuracy: 0.7812 - val_loss: 0.3935
Epoch 8/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 238ms/step - accuracy: 0.8653 - loss: 0.2936 - val_accuracy: 0.7292 - val_loss: 0.5097
Epoch 9/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 241ms/step - accuracy: 0.8897 - loss: 0.2822 - val_accuracy: 0.7500 - val_loss: 0.4351
Epoch 10/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 240ms/step - accuracy: 0.9201 - loss: 0.1777 - val_accuracy: 0.8542 - val_loss: 0.3510
Epoch 11/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 234ms/step - accuracy: 0.9488 - loss: 0.1881 - val_accuracy: 0.9479 - val_loss: 0.1822
Epoch 12/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 236ms/step - accuracy: 0.9653 - loss: 0.1211 - val_accuracy: 0.9583 - val_loss: 0.1692
Epoch 13/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 236ms/step - accuracy: 0.9382 - loss: 0.1674 - val_accuracy: 0.9167 - val_loss: 0.2066
Epoch 14/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 237ms/step - accuracy: 0.9806 - loss: 0.0679 - val_accuracy: 0.9062 - val_loss: 0.2202
Epoch 15/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 236ms/step - accuracy: 0.9673 - loss: 0.0810 - val_accuracy: 0.9583 - val_loss: 0.1458
Epoch 16/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 238ms/step - accuracy: 0.9947 - loss: 0.0394 - val_accuracy: 0.9167 - val_loss: 0.2043
Epoch 17/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 236ms/step - accuracy: 0.9954 - loss: 0.0396 - val_accuracy: 0.9896 - val_loss: 0.0735
Epoch 18/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 236ms/step - accuracy: 0.9923 - loss: 0.0347 - val_accuracy: 0.9479 - val_loss: 0.1280
Epoch 19/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 237ms/step - accuracy: 0.9994 - loss: 0.0095 - val_accuracy: 0.9896 - val_loss: 0.0344
Epoch 20/20
17/17 ━━━━━━━━━━━━━━━━━━━━ 4s 237ms/step - accuracy: 0.9956 - loss: 0.0124 - val_accuracy: 0.9792 - val_loss: 0.0727
```

Plotting the Accuracy

```
In [11]: plt.plot(history.history['accuracy'], color='blue', label='Training')
         plt.plot(history.history['val_accuracy'], color='red', label='Validation')
         plt.title('Accuracy')
         plt.xlabel("Epochs")
         plt.ylabel("Accuracy")
         plt.legend(loc='upper left')
         plt.show()
```



Sensitivity, Specificity and Accuracy

```
In [12]: from tensorflow.keras.metrics import TruePositives, TrueNegatives, FalsePositives, FalseNegatives, BinaryAccuracy
```

```
In [13]: true_positives = TruePositives()
         true_negatives = TrueNegatives()
         false_positives = FalsePositives()
         false_negatives = FalseNegatives()
         binary_accuracy = BinaryAccuracy()
```

```
In [14]: for batch in testing_set.as_numpy_iterator():
             X, y = batch
             yhat = model.predict(X)

             true_positives.update_state(y, yhat)
             true_negatives.update_state(y, yhat)
             false_positives.update_state(y, yhat)
             false_negatives.update_state(y, yhat)
             binary_accuracy.update_state(y, yhat)
```

```
In [15]: sensitivity = true_positives.result() / (true_positives.result() + false_negatives.result())
         specificity = true_negatives.result() / (true_negatives.result() + false_positives.result())

         print(f'Sensitivity: {sensitivity}, Specificity: {specificity}, Accuracy: {binary_accuracy.result()}')
```

```
Sensitivity: 0.9830508232116699, Specificity: 0.9900990128517151, Accuracy: 0.987500011920929
```