Dokuz Eylül University

Faculty of Engineering

Department of Computer Engineering

CME4409: Computer Graphics

Fall Term, 2024-2025

Final Project

1. Introduction

In this project, two different neural networks were developed to recognize hand gestures. The dataset, collected by the author, consists of two classes representing hand gestures for the numbers one and three. It includes a total of 220 images, making the dataset relatively small and the models susceptible to overfitting. Jupyter Notebook was used to execute the code and visualize the results. The first neural network is a simple Convolutional Neural Network (CNN), while the second is based on transfer learning using MobileNetV2.

The main objective of this project is to compare the performance of the two models and analyze their differences using test results. The comparison is based on metrics such as the confusion matrix, loss curves, accuracy, precision, and recall.

2. Description of Methods

In this project, the first step is to upload the data into Jupyter Notebook. The data is organized into two separate folders, each representing one class of images. After uploading, a few images and their corresponding true labels are displayed to confirm which class is associated with each label. In this model, label 1 represents class three, while label 0 represents class one. The images are then preprocessed to enhance the model's ability to recognize patterns. Preprocessing improves data clarity, which helps the computer perform more effectively. During this process, images are resized to 255x255 pixels to ensure uniformity in size and shape. The images are also converted from RGB to grayscale using TensorFlow’s rgb_to_grayscale function, which simplifies the image data and reduces computational complexity in some algorithms. Additionally, 10% of the images are rotated horizontally and vertically to diversify the dataset, exposing the model to a variety of variations. This step helps the model generalize better to unseen data during testing. Diversifying the data is especially crucial given the small size of the dataset, as it helps mitigate overfitting.

While designing the model the data needed to be separated to train, validation, and test. Using data.take() training data, size of 60% of the data, allocated to assigned batches. For validation and test, the batches that are already allocated are skipped using data.skip() and the leftover batches are allocated to validation and test.

The simple CNN model has two convolution layers, two max-pooling layers, one flatten layer, two fully connected layers (dense), and two dropout layers. The reason why the CNN model has very few layers is because the dataset the model is going to train is very small with only 220 images and two classes. More layers in the model will cause it to overfit the training data which means the data will not do well on the unseen data. The first convolution layer has 16 filters, each of size 2x2, with a stride of one meaning the filters move one pixel at a time. The activation function used in this layer is ReLu., which introduces non-linearity to the model. This layer is followed by a max-pooling layer with a pool size of (2x2). To prevent overfitting, 20% of the neurons are dropped using a dropout layer before passing to the next convolutional layer. The second convolution layer has 32 filters of size 2x2, also with a stride of one, and uses the ReLU activation function. This is followed by another max-pooling layer with a pool size of 2×2. After this, flatten layer is called. The intuition behind the flattening layer is to convert data into a 1-dimensional array preparing the data for the dense layers [1]. After these fully connected layers are used, a Fully Connected (FC) layer, aka a dense layer, is a type of layer where each neuron or node from the previous layer is connected to each neuron of the current layer [2]. The first fully connected layer contains 32 neurons in the dense layer, controlling the model's capacity to learn from the input data. And is using the ReLU activation function. This layer also adds L2 regularization (with a penalty factor of 0.01) to the weights (kernel) of the dense layer. This is used to prevent overfitting in the model. Additionally, 50% of the neurons are dropped using a dropout layer for regularization. The final dense layer has 1 neuron and uses the sigmoid activation function, which maps the output to a range of [0,1] making it ideal for binary classification. This allows the model to predict the probability of each input belonging to the positive class. The model is compiled with the Adam optimizer for efficient training, and the binary focal cross-entropy loss function to measure the match between predicted probabilities and true labels, with accuracy as the performance metric. This simple CNN architecture has over 4 million parameters, balancing complexity and the risk of overfitting given the dataset size.

To design the transfer learning model, the pre-trained MobileNetV2 model is used, excluding the fully connected layers at the top since the model is being customized for our specific task, only the convolutional and pooling layers from the pre-trained MobileNetV2 model are utilized, without the original classification layers. Next, we freeze the layers of the pre-trained model to preserve the features learned during training on ImageNet. However, to allow some flexibility in the model's learning, only the last 10 layers of the pre-trained model are unfrozen. This prevents overfitting by ensuring that the lower layers, which have learned basic features, remain unchanged while allowing the last few layers to adapt to the new task. The pre-trained MobileNetV2 model is then added to a new sequential model. Following this, a flatten layer is added to convert the 3D output from the convolutional layers into a 1D array suitable for input to fully connected layers. For the fully connected layers, the first dense layer has 32 neurons with a ReLU activation function. L2 regularization (with a penalty factor of 0.01) is applied to the weights of this layer to help prevent overfitting. Batch normalization is applied after this dense layer to normalize the input of the next layer and improve training stability. A dropout layer is applied before the second dense layer to randomly drop 20% of the neurons, which helps reduce overfitting. The second dense layer consists of 16 neurons and uses the same activation function and regularization as the first. Dropout is also applied here, with 20% of the neurons being dropped. Finally, the last dense layer consists of a single neuron with a sigmoid activation function, which is suitable for binary classification tasks. L2 regularization is again applied to this layer.

Both of the models were trained using the model.fit() method. The train data and validation data are added to this function and run with the epoch of 20. There are two callbacks added to this function: tf.keras.callbacks.TensorBoard() is used to visualize and save the log files, and EarlyStopping() is used to prevent overfitting, it follows the validation’s loss and if the loss won’t recover in the next 4 epochs it will stop the model training.

3. Results and comparisons

To begin comparing the models, the training and validation loss, as well as accuracy, are first compared between the CNN model and the transfer learning model. The CNN model's loss and accuracy graphs are shown below.

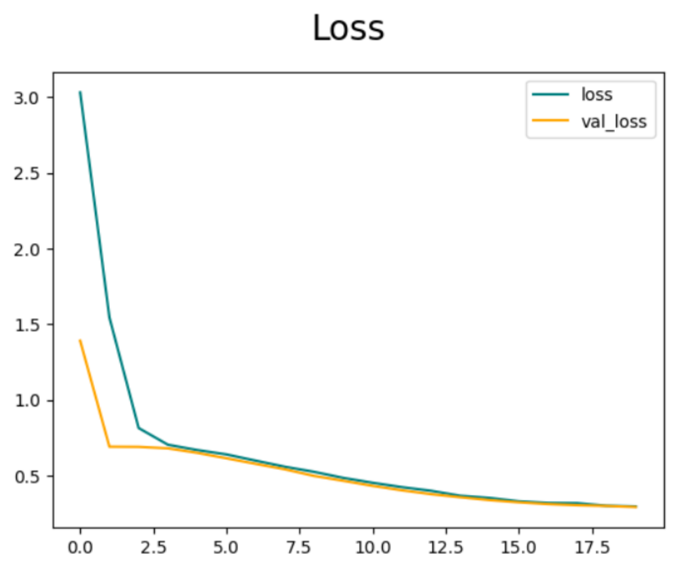


Figure 3.1: Loss values for the training and validation batch in the CNN model.

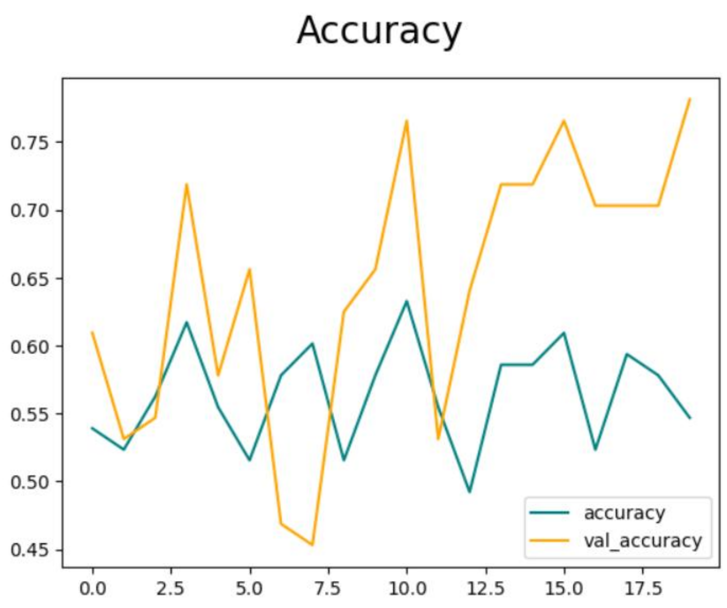


Figure 3.2: Accuracy values for the training and validation batch in the CNN model.

As can be seen from the image above the loos values for the train and validation are decreasing at a steady pace and we can't see sudden rises here. But in the accuracy, we see sudden rises for both train and validation. This is not something wanted since it means the model is not learning. When creating a model what we want is the loss of steadily decreasing while accuracy steadily increases. Sudden rises mean our model can be overfitting or underfitting.

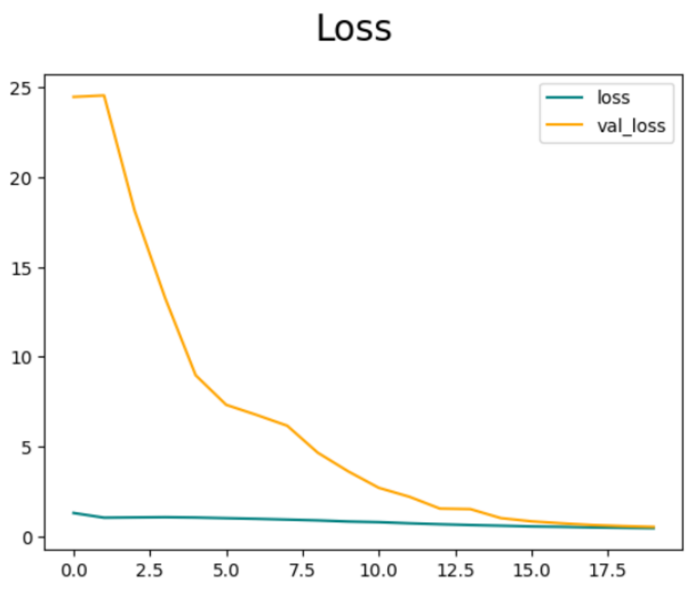


Figure 3.3: Loss values for the training and validation batch in the Transfer Learning model

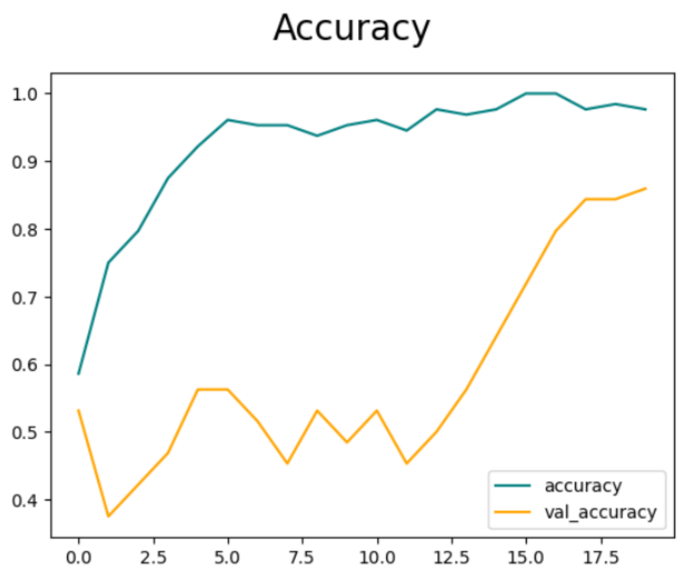
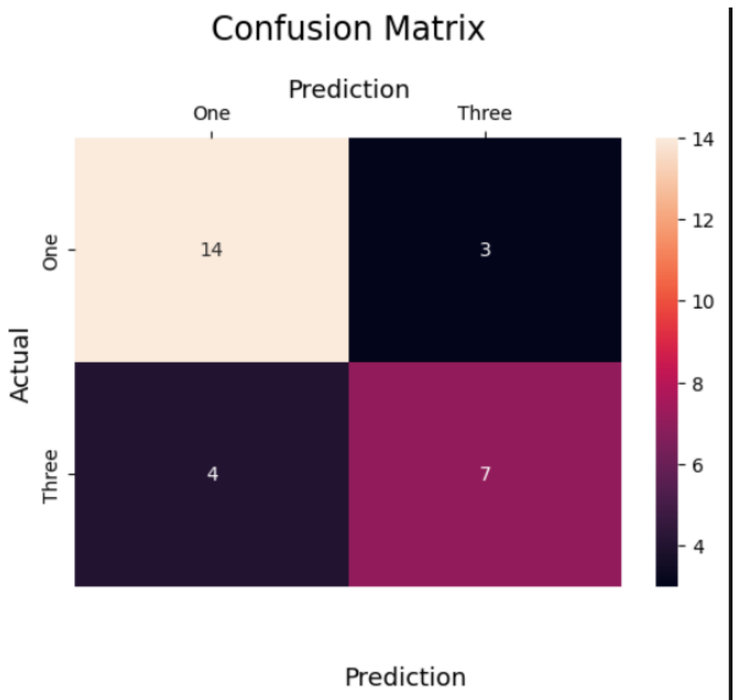
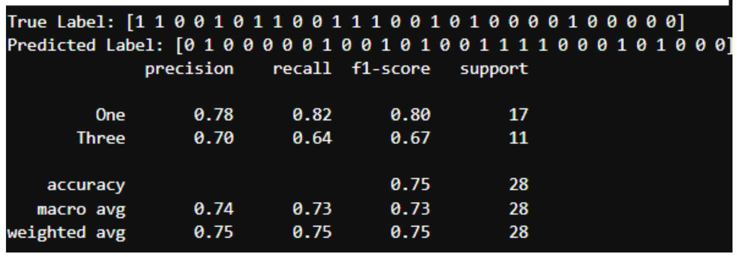


Figure 3.4: Accuracy values for the training and validation batch in the Transfer Learning model

As can be seen from the results, the transfer learning model demonstrates better performance in terms of both loss and accuracy. The loss decreases steadily without abrupt rises, indicating a stable learning process. Similarly, accuracy increases consistently for both training and validation. While there are minor spikes in the validation accuracy, these fluctuations are relatively small compared to those observed in the CNN model. Moreover, the transfer learning model corrects itself almost immediately, reflecting its ability to generalize and learn effectively. This behavior indicates that the transfer learning model is learning in a stable and consistent manner, leveraging pre-trained knowledge to achieve improved performance on the task.





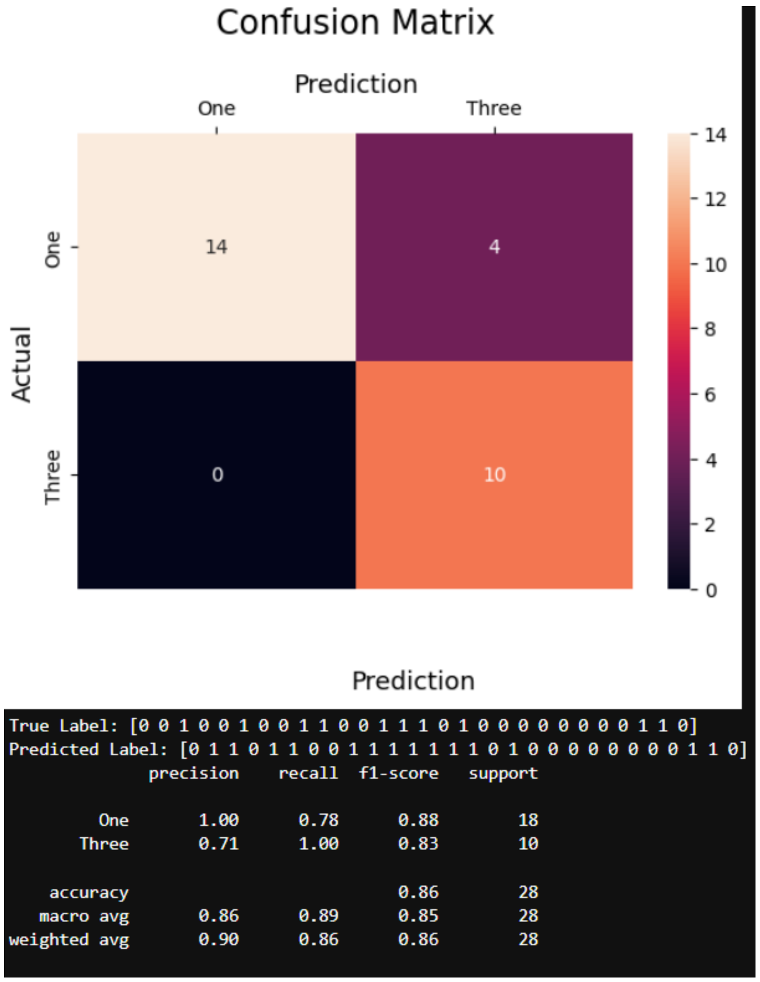
```
True Label: [1 1 0 0 1 0 1 1 0 0 1 1 1 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0]
Predicted Label: [0 1 0 0 0 0 0 1 0 0 1 0 1 0 0 1 1 1 1 0 0 0 1 0 1 0 0 0]
              precision    recall  f1-score   support

         One       0.78      0.82      0.80        17
       Three       0.70      0.64      0.67        11

    accuracy                           0.75        28
   macro avg       0.74      0.73      0.73        28
weighted avg       0.75      0.75      0.75        28
```

Figure 3.5: Confusion matrix of CNN model

Using the confusion matrix, we show how well a classification model is performing by comparing its predictions to the actual results. It breaks down the predictions into four categories: correct predictions for both classes (true positives and true negatives) and incorrect predictions (false positives and false negatives). Below the confusion matrix, we also show the precision, recal, and f-1 score. Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. Recall is the ratio of correctly predicted positive observations to all observations in the actual class. The F1-score is the harmonic mean of precision and recall. It balances the trade-off between precision and recall.

In the CNN model, the overall accuracy is 75%. Out of 17 images containing the label "One," 14 are correctly classified, while 3 are misclassified. Similarly, out of 11 images containing the label "Three," 7 are correctly classified, and 4 are misclassified. The precision for class "One" is 0.78, indicating that out of 18 images classified as "One," 14 were correctly identified. For class "Three," the precision is 0.70, meaning that out of 10 images classified as "Three," 7 were correctly identified.

These results suggest that the CNN model performs reasonably well and is mostly able to classify images correctly, with a slight imbalance in performance between the two classes. However, there is still room for improvement, particularly in reducing misclassifications for class "Three."



```
True Label: [0 0 1 0 0 1 0 0 1 1 0 0 1 1 1 0 1 0 0 0 0 0 0 0 0 1 1 0]
Predicted Label: [0 1 1 0 1 1 0 0 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 1 1 0]
              precision    recall  f1-score   support

         One       1.00      0.78      0.88        18
       Three       0.71      1.00      0.83        10

    accuracy                           0.86        28
   macro avg       0.86      0.89      0.85        28
weighted avg       0.90      0.86      0.86        28
```

Figure 3.6: Confusion matrix of Transfer Learning

In the Transfer Learning model, the overall accuracy is 86%. Out of 18 images containing the label "One," 14 are correctly classified, while 4 are misclassified. And out of 10 images containing the label "Three," all of them are correctly classified. The precision for class "One" is 1.0, indicating that out of 14 images classified as "One," all of them were correctly identified. For class "Three," the precision is 0.71, meaning that out of 14 images classified as "Three," only 10 of them were correctly identified.

When compared to the CNN model, the Transfer Learning model shows a significant improvement in overall accuracy, rising from 75% in the CNN model to 86%. This improvement highlights the advantages of leveraging pre-trained features in Transfer Learning. These results suggest that the Transfer Learning model is better at generalizing across both classes while maintaining consistency in classification, making it a more robust choice compared to the CNN model trained from scratch.

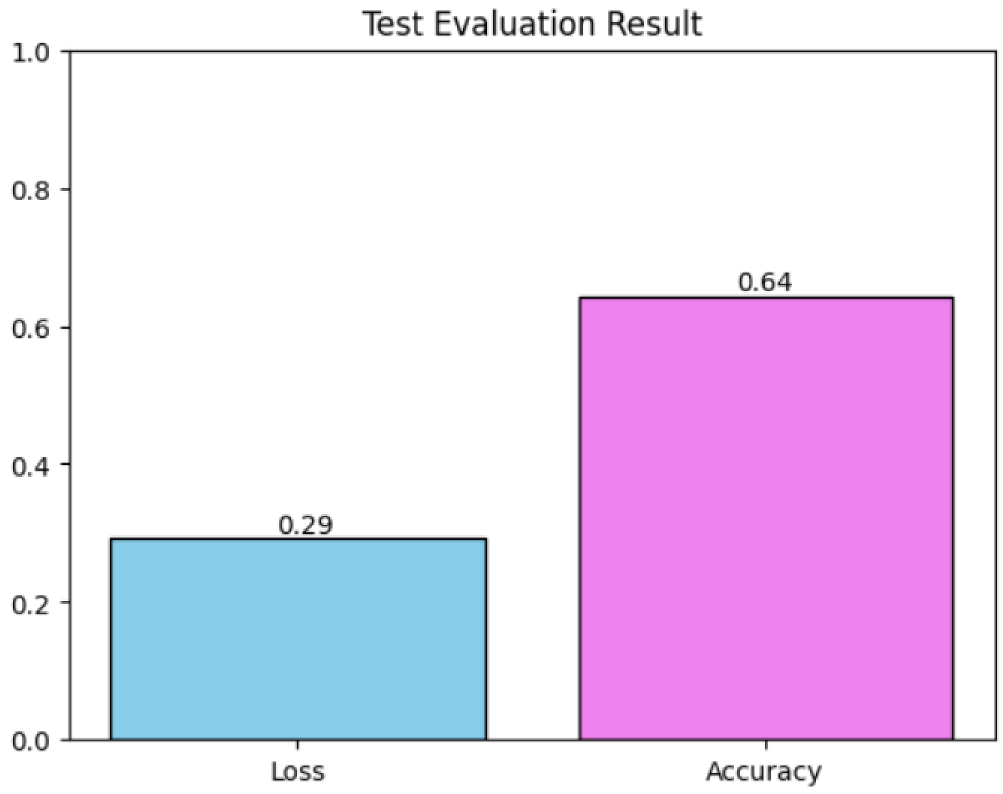


Figure 3.7: Loss and Accuracy of test results on the CNN model

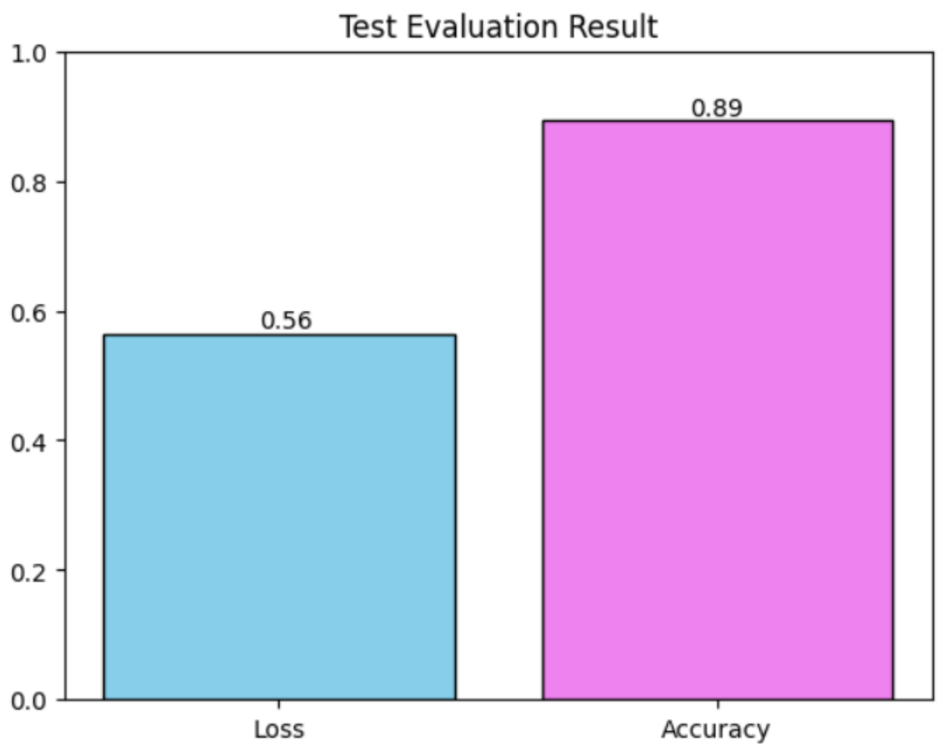


Figure 3.8: Loss and Accuracy of test results on the Transfer Learning model

As seen from the test results above, the Transfer Learning model achieves a significantly higher accuracy of 89% compared to the CNN model, which has an accuracy of only 64%. This difference in performance clearly shows that the Transfer Learning model is better at classifying images, likely due to the pre-trained features it leverages from a larger dataset. The pre-trained model can recognize more complex patterns and generalize better to new data, which is especially beneficial when training data is limited. However, while the Transfer Learning model outperforms the CNN model in terms of accuracy, it has a slightly higher test loss (0.56 vs. 0.29). This could be due to several factors. One possibility is that the Transfer Learning model, although it classifies images more accurately overall, might be slightly less confident in its predictions, leading to a higher loss. Another explanation could be that the CNN model is overfitting to the training data, as it has a lower loss but struggles to generalize well to unseen data, resulting in a lower accuracy. The CNN model may have learned to fit the training set very closely, but when evaluated on the test set, it struggles to generalize to new, unseen examples, leading to poorer performance.

4. Conclusion

In this project, two Convolutional Neural Networks (CNNs) were developed and compared using a relatively small dataset containing two classes. The results showed that the Transfer Learning model achieved a significantly higher accuracy on the test data, while the simple CNN model had a lower loss value. The CNN model, having a lower loss but also a lower accuracy, suggests that it might be overfitting the training data. On the other hand, the Transfer Learning model, which utilizes pre-trained weights from a larger dataset, demonstrated better classification accuracy. This indicates that transfer learning, with its ability to leverage knowledge from a larger and more diverse dataset, allowed the model to generalize better and classify images more accurately on the smaller dataset used in this project. While the transfer learning model had a slightly higher loss, the higher accuracy suggests that the model is effectively utilizing its pre-trained features to learn more relevant patterns and make better predictions on the test set.

In conclusion, the key takeaway from this comparison is that transfer learning, with its foundation in pre-trained networks, proved to be a more effective approach for classifying images in this case. The ability to train on a larger, more diverse dataset gave the transfer learning model a significant advantage over the simpler CNN model, resulting in better generalization and accuracy.

5. Resources

Ali, M., S. (2022, June 23). Flattening CNN layers for Neural Network and basic concepts. Medium. https://medium.com/@muhammadshoaibali/flattening-cnn-layers-for-neural-network-694a232eda6a

Geeks for Geeks. (2024, May 18). Fully Connected Layer vs Convolutional Layer. https://www.geeksforgeeks.org/fully-connected-layer-vs-convolutional-layer/?ysclid=m6atlzftx1240693437