Transfer Learning in Image Classification

Final Project Report

INFO 536 – Applied Machine learning

By

Muhammad Qureshi	**B00980214**
S V Arun Varma Vanaparthi	**B00977899**
Nitish Devineni	**B00974880**
Loksharan Saravanan	**B00979751**

Summary

The "Transfer Learning" project focuses on using transfer learning techniques in image classification, which might be useful in situations when there isn't much data. The project is based on the concepts of object-oriented programming (OOP) and uses two main models: a transfer learning model that uses a pre-trained network and a base model that is created from scratch.

TensorFlow and Keras are used to create the base model, which has many convolutional layers, max pooling, dropout for regularization, and dense layers. The augmented dataset, which is improved with different image transformations like rotation, shifting, shear, zooming, and flipping, is especially used to train this model. This method ensures a strong training process while addressing the issue of limited data.

The project uses a pre-trained model (FModel) in parallel and modifies it for the MNIST digit classification task. To preserve learnt features, the layers of the pre-trained model are frozen up to the Flatten layer, and new dense layers are added specifically for the task of digit recognition. Using a scaled and normalized dataset, this model is trained to demonstrate the effectiveness of transfer learning in adjusting to new tasks.

Both models go through a rigorous training and validation process, and accuracy and loss measures are used to assess how well they perform. A balance between training and validation metrics for the base model and noteworthy accuracy for the transfer learning model is indicated by the graphical representations of these metrics, which offer insights into the learning curves of the models.

Introduction & Proposal

Overview:
The use of image augmentation and transfer learning techniques to picture classification tasks is the main emphasis of this study. The goal is to illustrate how these techniques can greatly improve model performance and show their effectiveness when data is limited.

Project Objective:
The project aims to categorize photographs into several categories using a pre-trained model that has been updated and fine-tuned for the specific task. In addition, the initiative intends to increase the datasets to overcome the limits of limited datasets.

Scope and Relevance:
This method is especially useful in domains where gathering huge datasets is difficult or impractical. The research demonstrates how high picture classification accuracy may be attained with less data by utilizing transfer learning and image augmentation.

Methodology

a. Network Structure

Pre-trained Model Utilization: The project makes use of a pre-trained model that is subsequently adjusted for the categorization task. Layers are added or removed from the pre-trained model's structure to better fit the new assignment.

Image Augmentation: The project uses picture augmentation techniques to solve the problem of limited data. This involves using transformations like rotation, width and height shifting, shear mapping, zooming, horizontal flipping, and nearest fill mode to create new images.

Explanation of the deep learning Model components:

Building Base model:

Convolutional Layers (Conv2D)
Function: Convolutional layers are the core building blocks of a Convolutional Neural Network (CNN). To extract features, they filter the input data through a convolution procedure. These layers work especially well when processing data that has a topology resembling a grid, like photographs.
Usage in Model: Multiple convolutional layers with different numbers of filters (32, 64, 128, 256, 512, 1024, 2048) are used in the model. Higher-level features are gradually extracted from the input image by each layer. The Rectified Linear Unit (ReLU) activation function, which adds non-linearity to the model and enables it to learn more intricate patterns, is indicated by the activation='relu'.

Max Pooling Layers (MaxPooling2D)
Function: Max pooling minimizes the input volume's width and height. It aids in lowering the computation and parameter count in the network, which helps to prevent overfitting.
Use in Model: These layers are used after specific convolutional layers, effectively downsampling the feature maps and lowering their dimensionality for computational efficiency and to extract dominating features.

Flatten Layer
Function: The flattened layer creates a 1D feature vector from the 2D feature maps obtained from the convolutional layers. Convolutional layers must go through this process to become fully connected layers, or dense layers.
Usage in Model: It appears only once in the model, acting as a link between the convolutional and dense layers.

Dense Layers
Function: Dense layers, also known as fully connected layers, perform classification on the features retrieved by convolutional layers. A dense layer is "fully connected" when every neuron in it receives input from every other layer's neuron.
Use in Model: ReLU-activated neurons in multiple dense layers (1024, 512, 256, 128, 64) are used in the model. The last dense layer uses a softmax activation function for multi-class classification and has as many neurons as there are classes in the dataset.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling (Rescaling)        (None, 90, 120, 1)        0
_________________________________________________________________
conv2d (Conv2D)              (None, 90, 120, 32)       320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 90, 120, 64)       18496
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 45, 60, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 45, 60, 128)       73856
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 22, 30, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 22, 30, 256)       295168
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 11, 15, 256)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 11, 15, 512)       1180160
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 7, 512)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 5, 7, 1024)        4719616
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 2, 3, 1024)        0
_________________________________________________________________
dropout (Dropout)            (None, 2, 3, 1024)        0
```

```
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 2, 3, 2048)        18876416
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 1, 1, 2048)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 1, 2048)        0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 1024)              2098176
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_3 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_5 (Dense)              (None, 47)                3055
=================================================================
Total params: 27,962,543
Trainable params: 27,962,543
Non-trainable params: 0
```

Building Model based on pretrained model: Transfer Learning:

Pre-trained Model Adaptation

Loading and Adapting Pre-trained Model: The pre-trained model (‘FModel’) is loaded, and its layers are used up to the ‘Flatten; layer. To accurately slice the model, the index of the Flatten layer is found.
Freezing Layers: The weights of these layers are frozen (‘layer.trainable = False’) to preserve the learned features. This implies that the weights of these layers won't be changed while the new model is being trained.

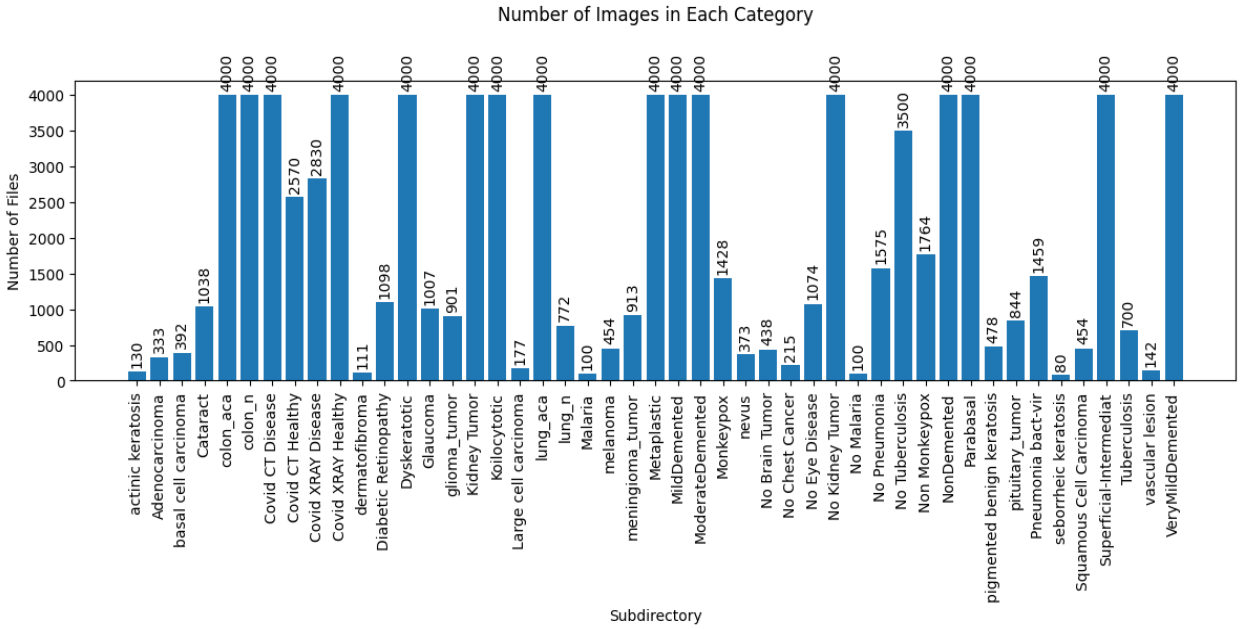
Dense Layers: After the layers of the pre-trained model, new dense (completely linked) layers are added. The purpose of these layers is to categorize the numbers in the MNIST dataset. The model becomes non-linear as the number of neurons with relu activation increases (512, 256, 128, 64, 32) in each dense layer.
Output: The last layer comprises 10 neurons, which are equivalent to the 10 classes (digits 0-9) present in the MNIST dataset. Here, the typical activation function for multi-class classification tasks—the softmax activation function—is applied.

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_9 (Sequential)    (None, 2048)              25164032
_________________________________________________________________
dense_30 (Dense)             (None, 512)               1049088
_________________________________________________________________
dense_31 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_32 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_33 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_34 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_35 (Dense)             (None, 10)                330
=================================================================
Total params: 26,388,010
Trainable params: 1,223,978
Non-trainable params: 25,164,032
```

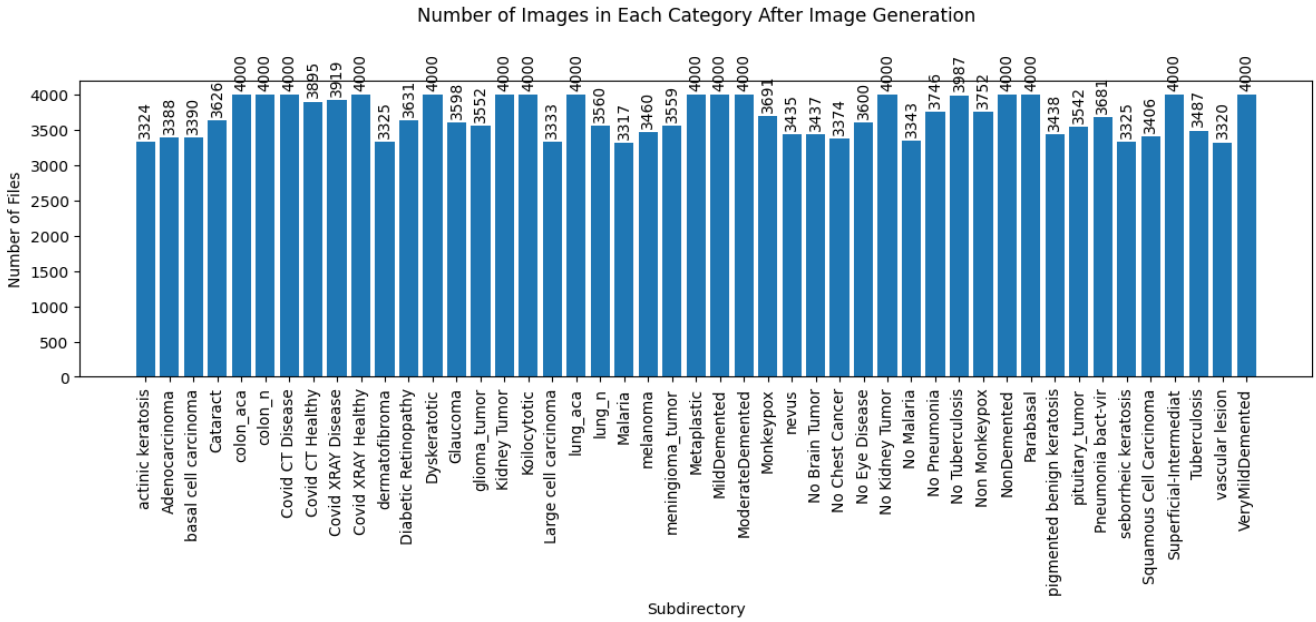
b. Training & Validation Process

Data Preparation: The dataset, which was initially small, is expanded to generate a more robust collection for training. Preprocessing, resizing, and category format encoding of the labels are applied to the images.
Training Strategy: To achieve generalization and prevent overfitting, the model is trained on the augmented dataset with a split between training and validation data.

Number of images generated:



Number of images generated between 3000-4000:



Evaluation & Results

a. Training & Validation Results

Performance Metrics: Accuracy and loss metrics are used to assess the model's performance. To illustrate the learning process, training and validation accuracies and losses are displayed over epochs.

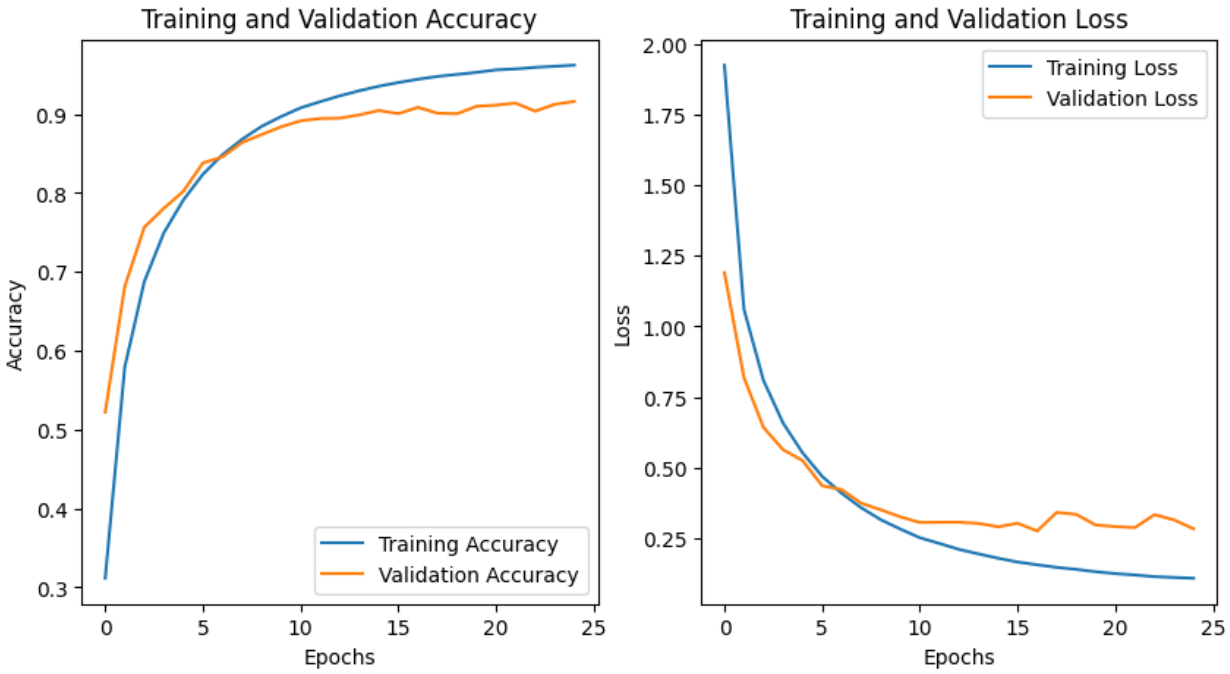
Result Analysis:

The convergence behavior and learning efficiency of the model are revealed by examining the learning curves (accuracy and loss) over training epochs.

Compilation and Training

- Optimizer: The Adam optimizer is employed, which is a stochastic gradient descent extension that is well-known for its effectiveness in dealing with sparse gradients and adaptive learning rates.
- Loss Function: The model, which is appropriate for multi-class classification applications, has 'categorical_crossentropy' as its loss function.
- Training Procedure: Using both training and validation datasets, the model is trained for 25 epochs with a batch size of 16. To minimize the loss function and progressively enhance the model's performance, this procedure entails modifying the model weights.

Model Evaluation
Test Accuracy: After training, the model's performance is evaluated on the test dataset to measure its accuracy, which is the proportion of correctly predicted instances among the total instances evaluated.



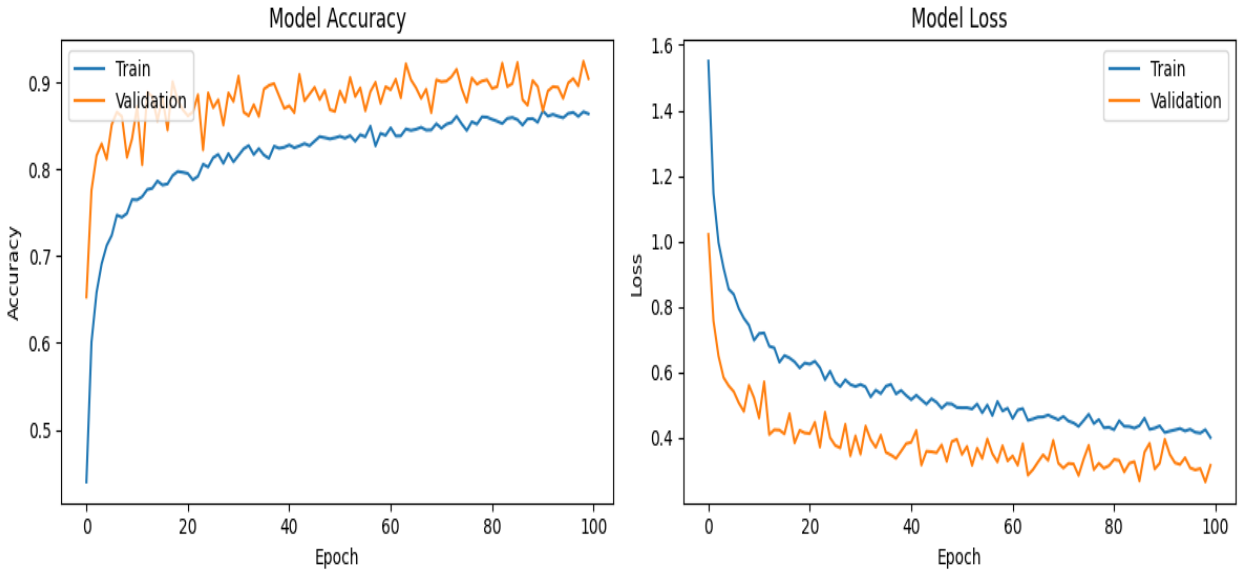
Training and validation

b. Performance Compared to Baselines

Baseline Comparisons: The adjusted model's performance is compared to that of other models or network topologies. This comparison aids in evaluating the suitability of the selected architecture and the transfer learning strategy.

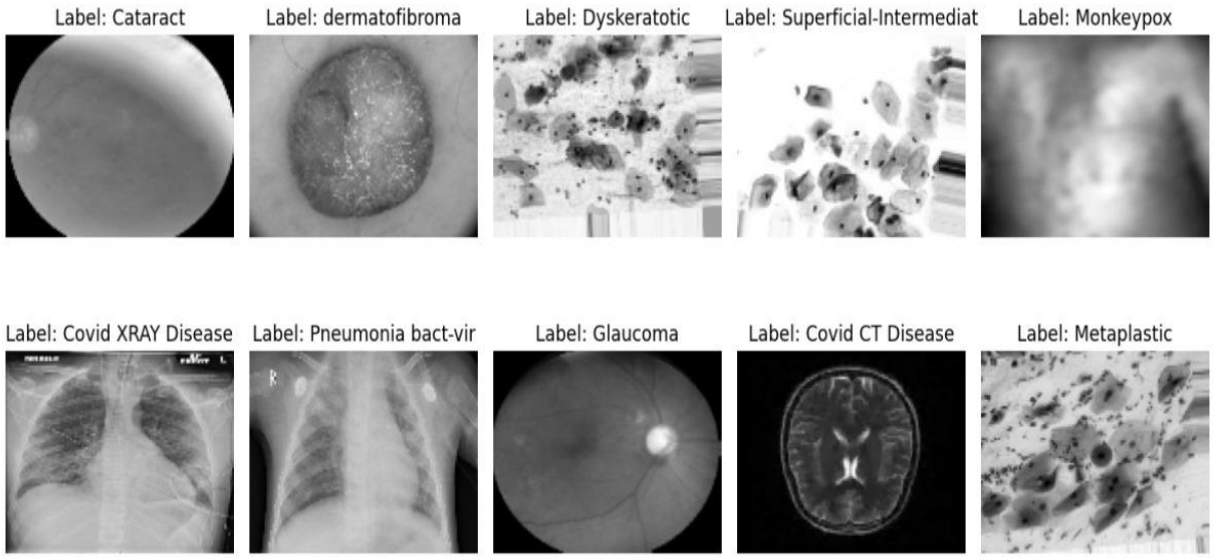
Model Compilation and Training
Compilation: The Adam optimizer and the categorical_crossentropy loss function are used to compile the model. For multi-class classification tasks, this configuration is common.
Training Procedure: Using both training and validation datasets from MNIST, the model is trained for 100 epochs with an 8-batch size. Through this procedure, the pre-trained model's feature extraction skills are utilized, enabling the new layers to gain knowledge from the MNIST data.

Model Evaluation
Test Accuracy: The model's performance is evaluated on a separate test set, and the test accuracy is reported. This metric provides insight into how well the model generalizes to unseen data.



c. Hyperparameter Tuning

Hyperparameter Adjustments: Details of hyperparameter tuning are presented, such as learning rate adjustments, dropout rates, and activation functions, are provided. The performance of the model is examined in relation to these parameters.



Images that are generated

Conclusion

Key Findings: With little data, the study effectively illustrates the power of transfer learning in conjunction with picture augmentation for image classification tasks. When a pre-trained model is used, training time and computational resources are greatly decreased without sacrificing accuracy.

Future Directions: Several pre-trained model experiments, an examination of alternative picture augmentation methods, and the application of the methodology to diverse other kinds of data are some ideas for future research.

The project's approach, methods, and conclusions are summarized in this article, which also highlights the potential benefits of picture augmentation and transfer learning for improving the performance of image classification models, particularly in settings with little data.