ML Project

UCS1603 Introduction to Machine Learning

PROJECT TEAM MEMBERS

NIVEDHITHA D
185001104

LAKSHMI PRIYA B
185001083

RAGHAV R
185001119

SAKTHI SAIRAJ
185001134

CONTENTS

Problem Statement

01

Brief discussion of the problem statement & proposed solution

POTHOLE & PLAIN ROAD CLASSIFICATION
Using
TRANSFER LEARNING
&
CNN

Problem Statement

Using deep learning and transfer learning techniques to solve the binary image classification problem of separating plain roads and roads with potholes by adopting an accurate model for savings in training time and computational efficiency.

Pothole

Plain road

Dataset Details

02

Exploration of the dataset

Key Dataset Details

➢ NAME	: Pothole and Plain Road Images
➢ SOURCE	: Kaggle
➢ RECORD TYPE	: .jpg, .jpeg , .png, .gif images
➢ CLASSES	: Binary Classification - pothole and plain
➢ NUMBER OF RECORDS	: 700 (350 each)
➢ TRAIN-VAL-TEST SPLIT	: 60-20-20 ratio (210-70-70)
➢ FEATURES	: Convolutional Base (Deep Learning)

Dataset Stats

```
In [10]:
# Sanity checks
print('total training plain images :', len(os.listdir(train_plain_dir)))
print('total training pothole images : ',len(os.listdir(train_pothole_dir)))
print('total validation plain images :', len(os.listdir(validation_plain_dir)))
print('total validation pothole images :', len(os.listdir(validation_pothole_dir)))
print('total test plain images :', len(os.listdir(test_plain_dir)))
print('total test pothole images :', len(os.listdir(test_pothole_dir)))
```

```
total training plain images : 210
total training pothole images : 210
total validation plain images : 70
total validation pothole images : 70
total test plain images : 70
total test pothole images : 70
```

Directory Structure

```
admin@X507UAR:~/Desktop/Pothole-Classification-main$ tree -Ld 3 .
.
├── Code
├── Data
│   ├── test
│   │   ├── plain
│   │   └── pothole
│   ├── train
│   │   ├── plain
│   │   └── pothole
│   └── validation
│       ├── plain
│       └── pothole
└── Results

12 directories
admin@X507UAR:~/Desktop/Pothole-Classification-main$
```

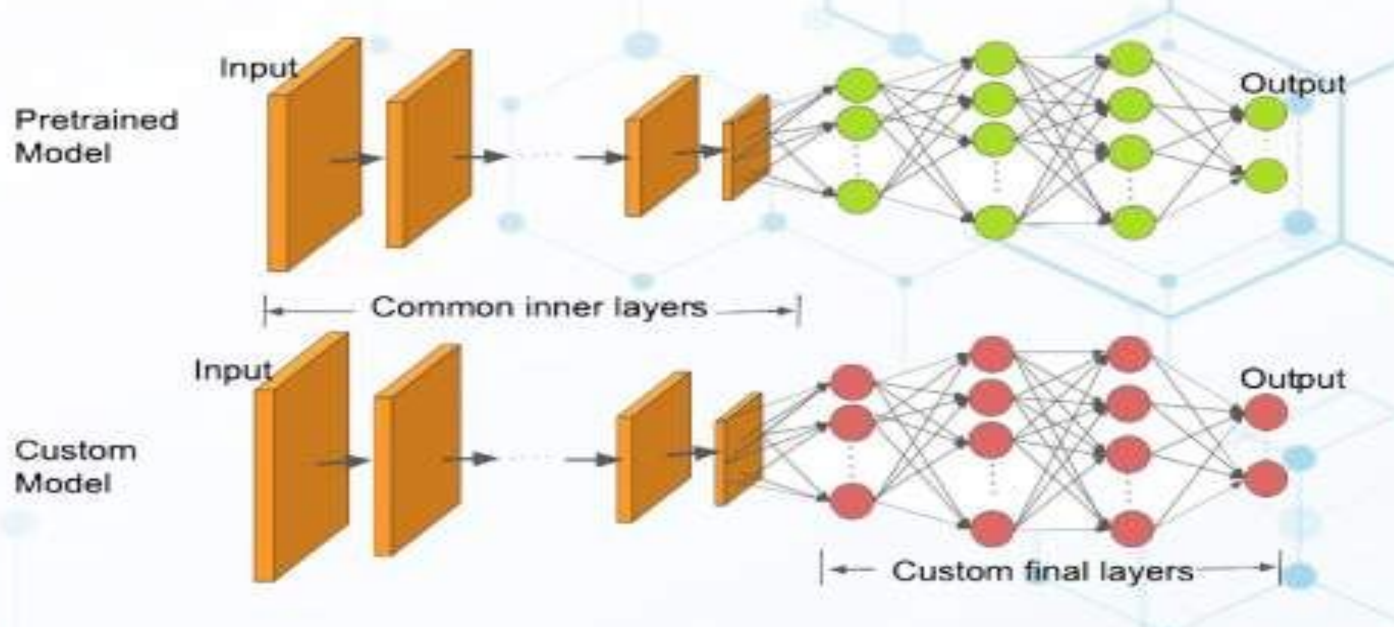
Methodology 03

Step-by-step approach to solve the problem and tentative workflow

Transfer Learning

- Transfer learning is a popular method in computer vision because it helps **build accurate models with significant savings in time**
- With transfer learning, we can solve problems from patterns that have already been learned when solving a different problem
- This way we can leverage previous learnings and avoid starting from scratch
- A pre-trained model is a model that was trained on a large benchmark dataset to solve a problem similar to the one that we want to solve
- Due to the computational cost of training such models, it is common practice to import and use models from published literature (e.g. VGG, Inception, MobileNet)

Transfer Learning



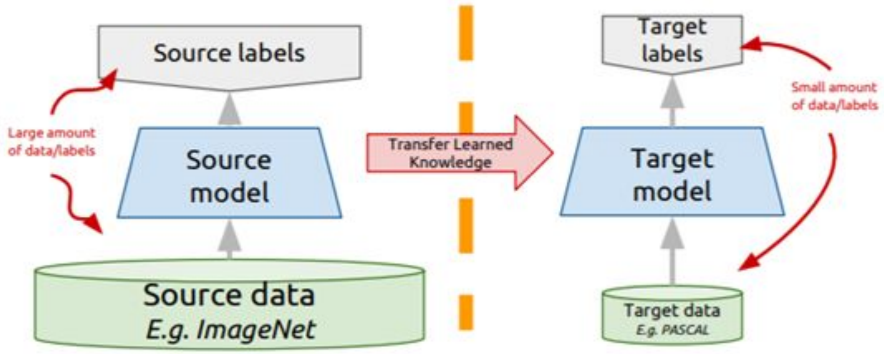
Transfer learning: idea

Instead of training a deep network from scratch for your task:

- Take a network trained on a different domain for a different **source task**
- Adapt it for your domain and your **target task**

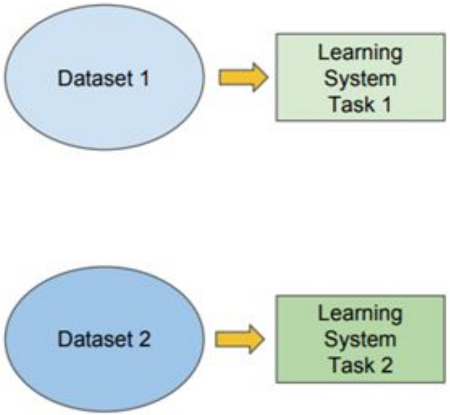
Variations:

- Same domain, different task
- Different domain, same task

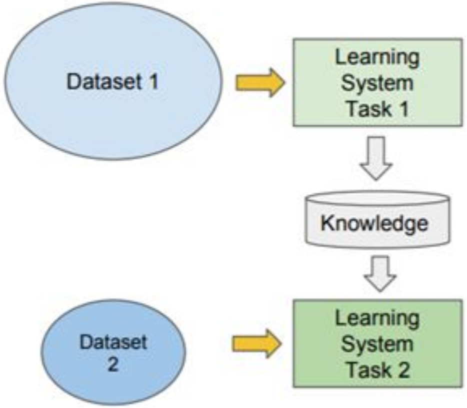


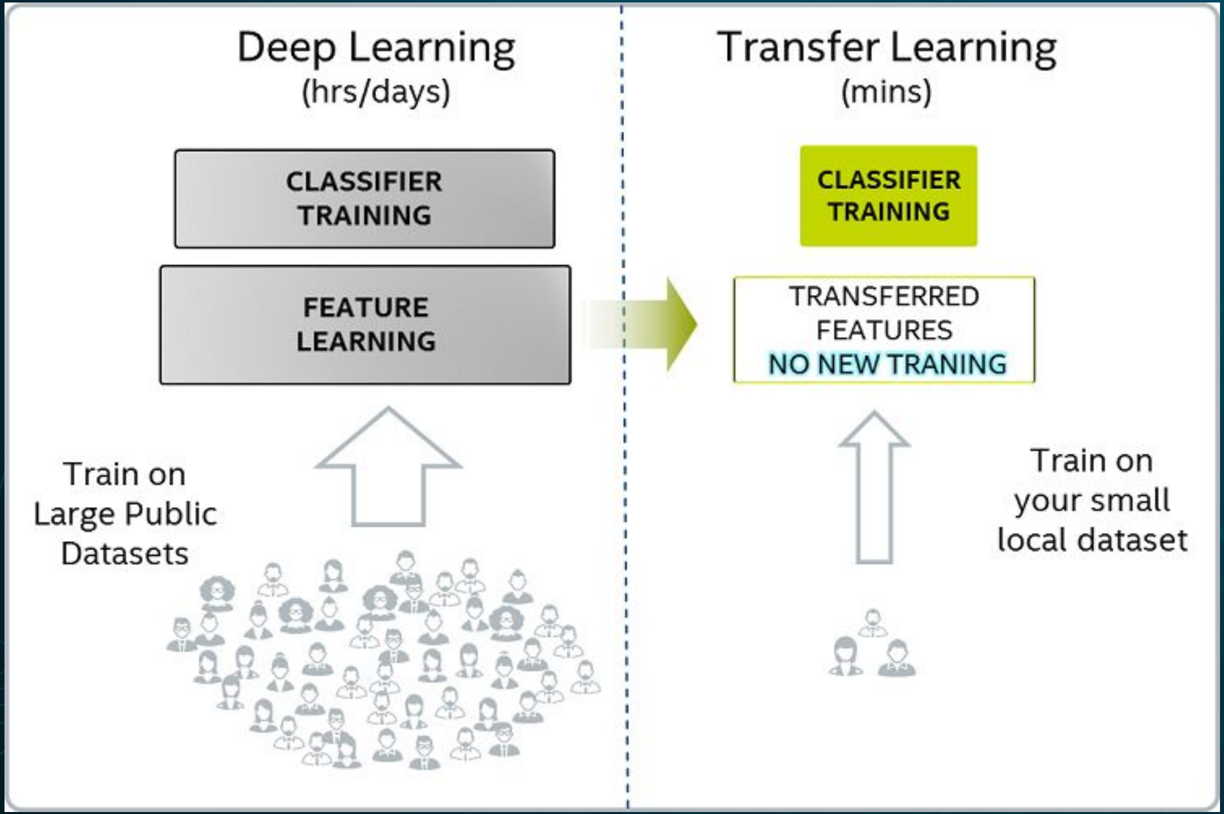
Traditional ML vs Transfer Learning

- Isolated, single task learning:
  - Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks



- Learning of a new tasks relies on the previous learned tasks:
  - Learning process can be faster, more accurate and/or need less training data

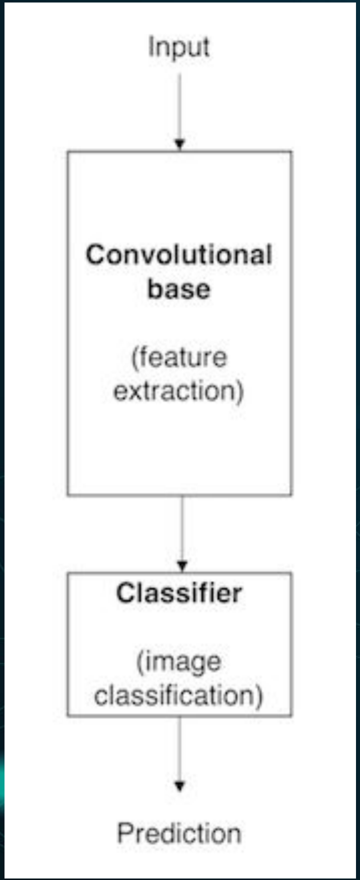



Deep Learning
(hrs/days)
CLASSIFIER TRAINING
FEATURE LEARNING
Train on Large Public Datasets
Transfer Learning
(mins)
CLASSIFIER TRAINING
TRANSFERRED FEATURES
NO NEW TRANING
Train on your small local dataset

Convolutional Neural Networks

- Several pre-trained models that are used in transfer learning are based on large CNNs
- **High performance** and **ease** of training are two of the main factors driving the popularity of CNN over the last years
- A typical CNN has two parts:
  - **Convolutional base**, which is composed by a stack of convolutional and pooling layers with the main goal of generating features from the image
  - **Classifier**, which is usually composed by fully connected layers (layer whose neurons have full connections to all activation in the previous layer), has the main goal of classifying the image based on the detected features

CNN Architecture



- This deep learning model can automatically learn **hierarchical feature representations**
- This means that **features computed by the first layer are general and can be reused in different problem domains, while features computed by the last layer are specific and depend on the chosen dataset and task**
- *'If first-layer features are general and last-layer features are specific, then there must be a transition from general to specific somewhere in the network'*
- As a result, the convolutional base of CNN — especially its lower layers (those who are closer to the inputs) — refer to general features, whereas the classifier part, and some of the higher layers of the convolutional base, refer to specialised features

Repurposing a pre-trained model

To repurpose a pre-trained model for our own needs, we start by removing the original classifier, then we add a new classifier that fits our purposes, and finally we have to **fine-tune our model according to one of three strategies**:

Train entire model

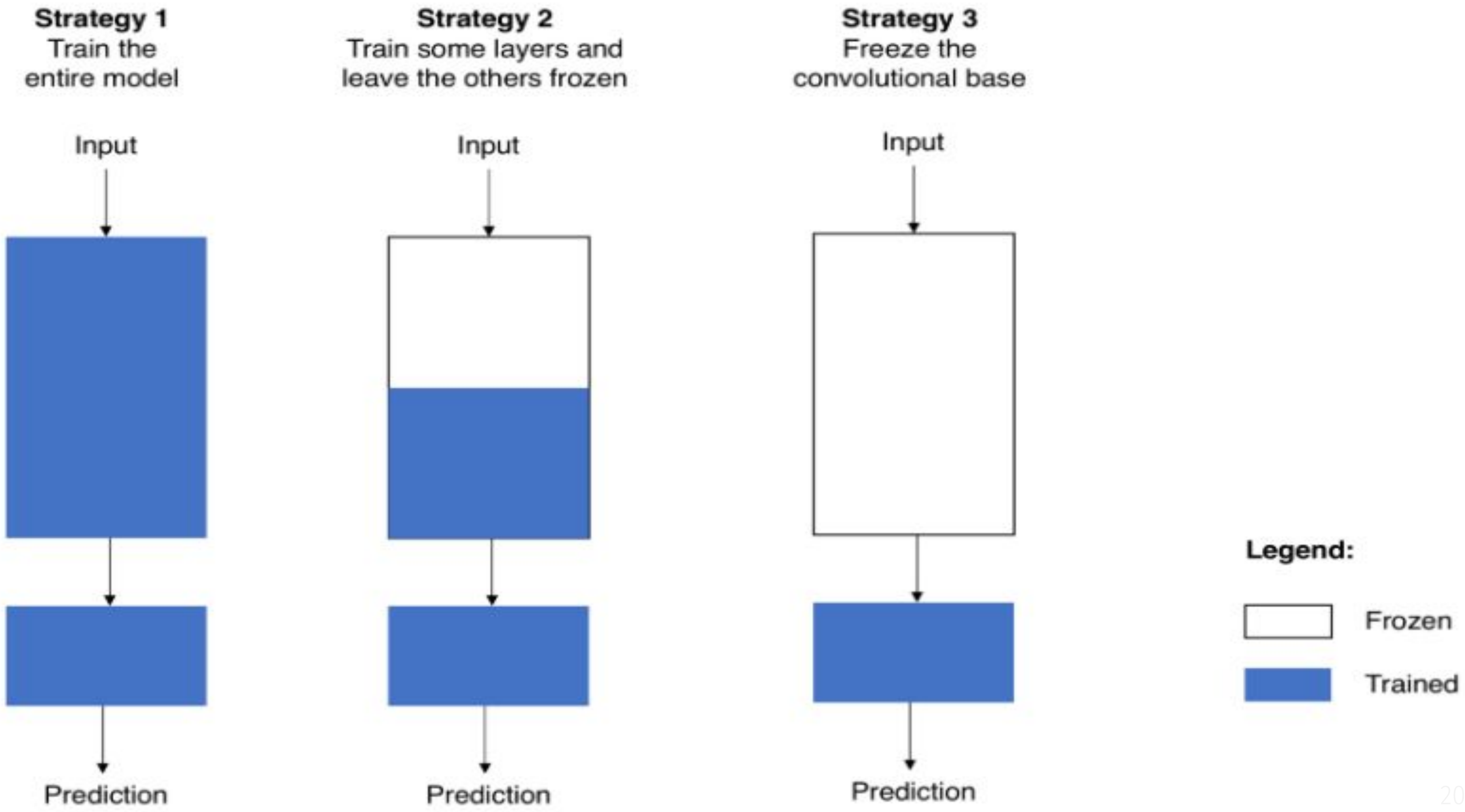
- Learning the model from scratch
- Needs a large dataset
- Lot of computational power

Train some layers & leave others frozen

- Lower layers refer to general features (problem independent), while higher layers refer to specific features (problem dependent)
- Here, we play with that dichotomy by choosing how much we want to adjust the weights of network (a frozen layer does not change during training)

Freeze convolutional base

- This case corresponds to an extreme situation of train/freeze trade-off
- The main idea is to keep the convolutional base in its original form and then use its outputs to feed the classifier

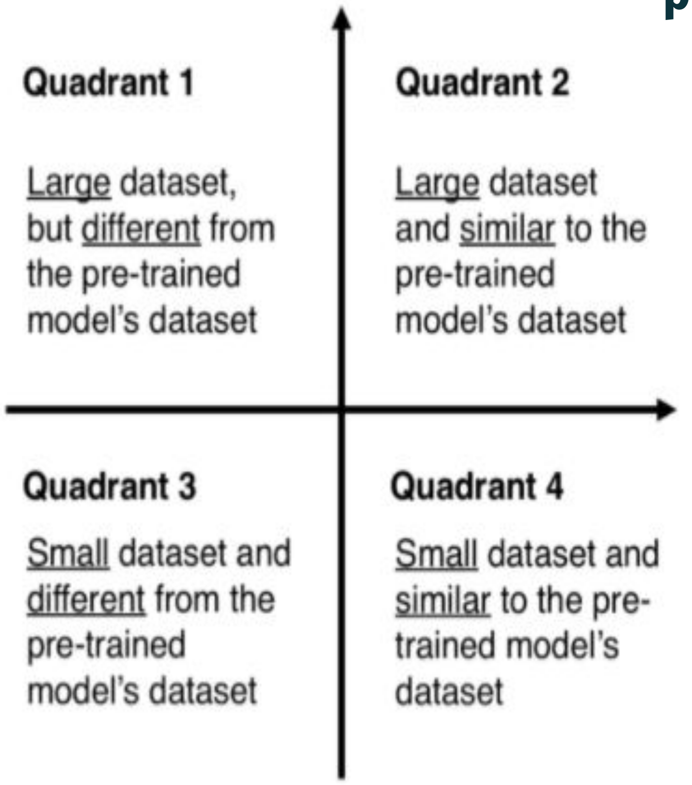

Strategy 1
Train the entire model
Input
Prediction
Strategy 2
Train some layers and leave the others frozen
Input
Prediction
Strategy 3
Freeze the convolutional base
Input
Prediction
Legend:
Frozen
Trained

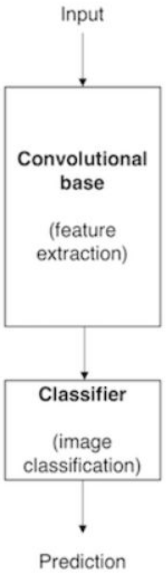
Transfer Learning Process

1. Select a pre-trained model based on the problem statement
2. Classify your problem according to the Size-Similarity Matrix
    a. classifies your computer vision problem considering the size of your dataset and its similarity to the dataset in which your pre-trained model was trained
    b. As a rule of thumb, consider that your dataset is small if it has less than 1000 images per class
3. Fine-tune your model using the Size-Similarity Matrix to guide your choice and then refer to the three options mentioned before about repurposing a pre-trained model

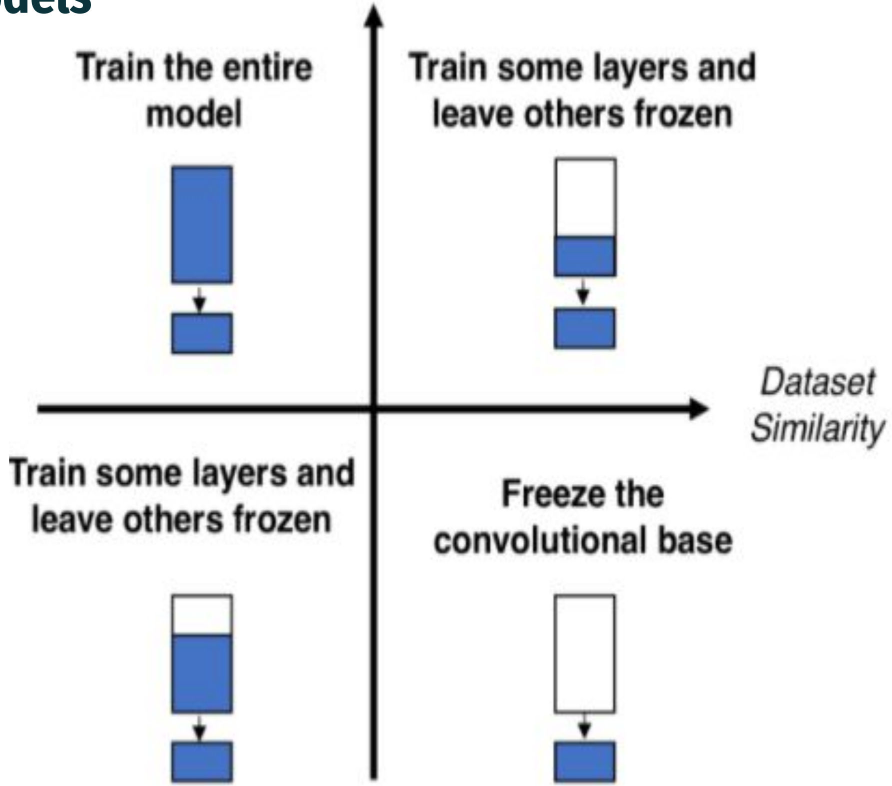
Size-Similarity Matrix and Decision Map for fine-tuning pre-trained models

Dataset Size





Dataset Size



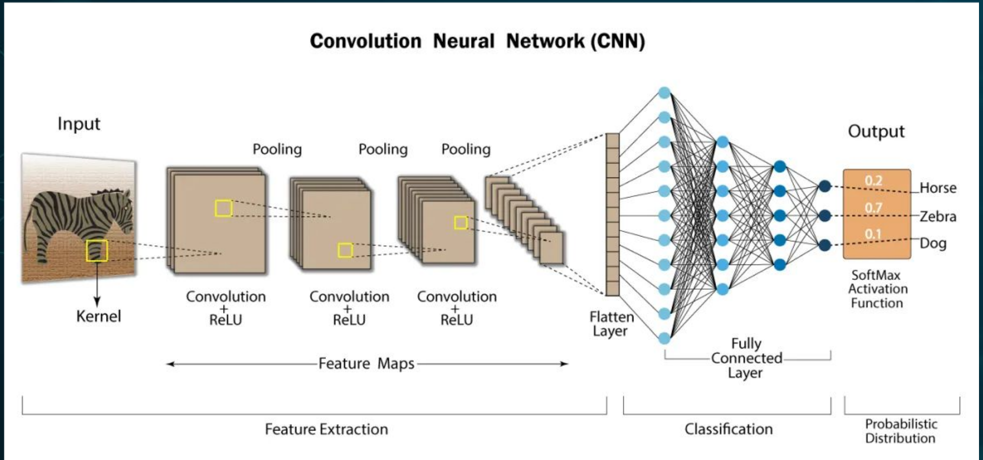
Classifiers on top of deep CNNs

Models for image classification that result from a transfer learning approach based on pre-trained convolutional neural networks are usually composed of two parts:

1. Convolutional base, which performs feature extraction
2. Classifier, which classifies the input image based on the features extracted by the convolutional base

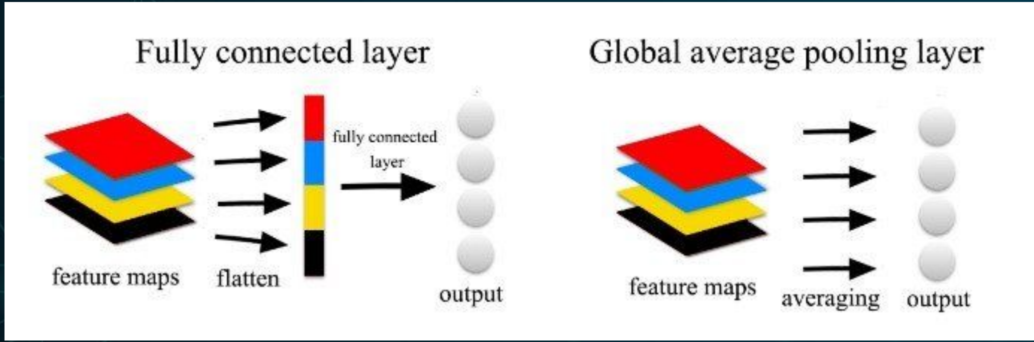
Fully-connected Layers

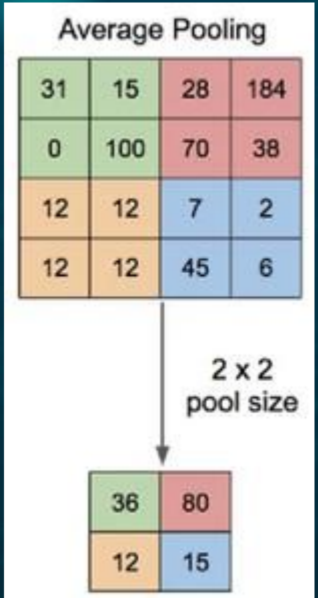
- For image classification problems, the standard approach is to use a stack of fully-connected layers followed by a softmax activated layer
- The softmax layer outputs the probability distribution over each possible class label and then we just need to classify the image according to the most probable class



Global Average Pooling

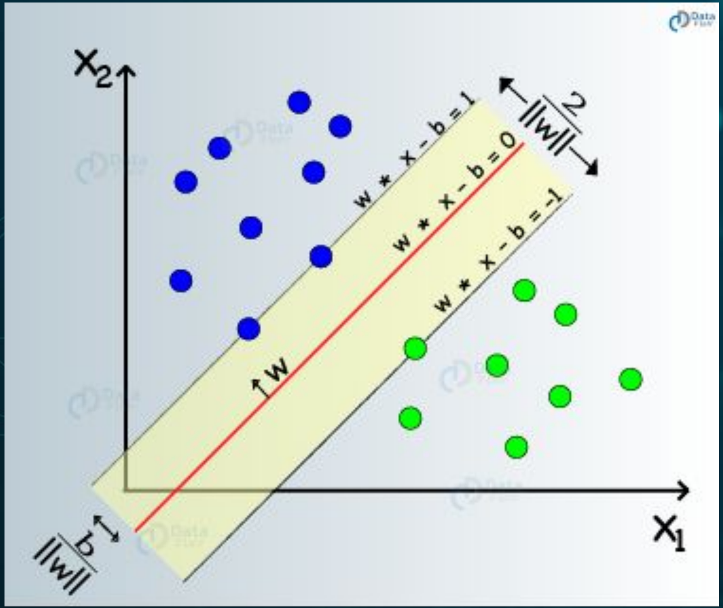
Instead of adding fully connected layers on top of the convolutional base, we add a global average pooling layer and feed its output directly into the softmax activated layer.





Linear Support Vector Machines

We can improve classification accuracy by training a linear SVM classifier on the features extracted by the convolutional base



Proposed Workflow

PREPARE DATA

Choose appropriate dataset. A smaller version of the original dataset can be used to run the models faster, which is great for people who have limited computational power

EXTRACT FEATURES

Perform feature extraction from convolutional base. These features will feed the classifiers which we want to train

Proposed Workflow

BUILDING CLASSIFIERS

1. **Fully-connected layers** - This classifier adds a stack of fully-connected layers that is fed by the features extracted from the convolutional base.

2. **Global average pooling** - The difference between this case and the previous one is that, instead of adding a stack of fully-connected layers, we will add a global average pooling layer and feed its output into a sigmoid activated layer.

3. **Linear support vector machines** - In this case, we will train a linear support vector machines (SVM) classifier on the features extracted by the convolutional base. We will use k-fold cross-validation to estimate the error of the classifier. Since k-fold cross-validation will be used, we can concatenate the train and the validation sets to enlarge our training data (we keep the test set untouched, as we did in previous cases).

Proposed Workflow

TRAIN MODEL

Train the model by fixing the epochs and batch size

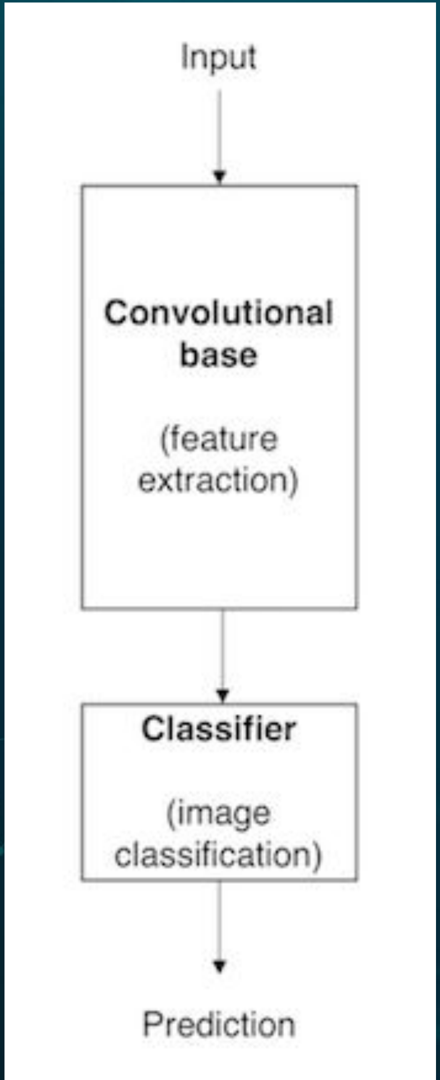
PERFORMANCE ANALYSIS

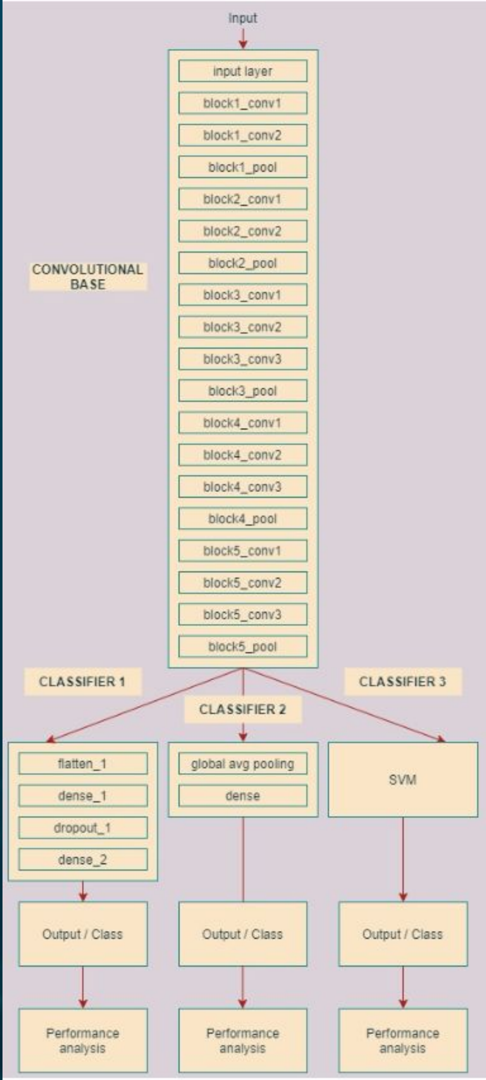
Analyse the performance of the model using various performance metrics

Model Architecture 04

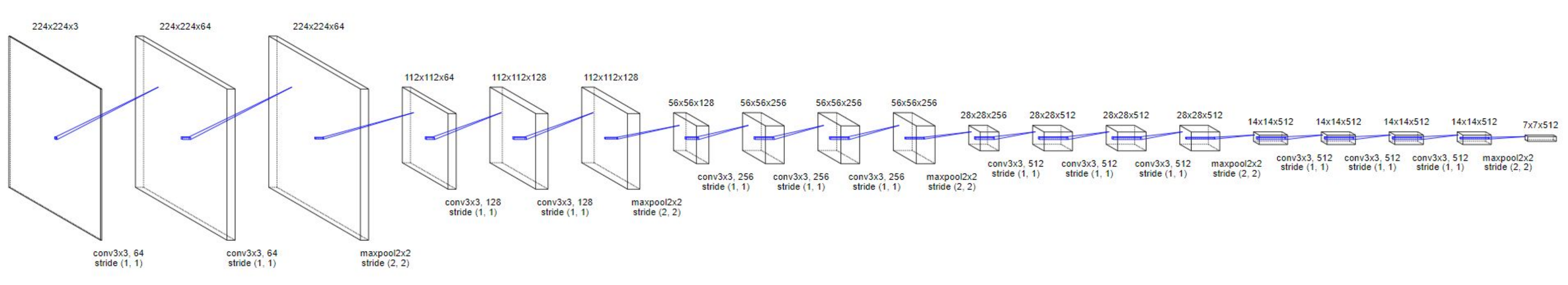
Architecture of the model built

Architecture

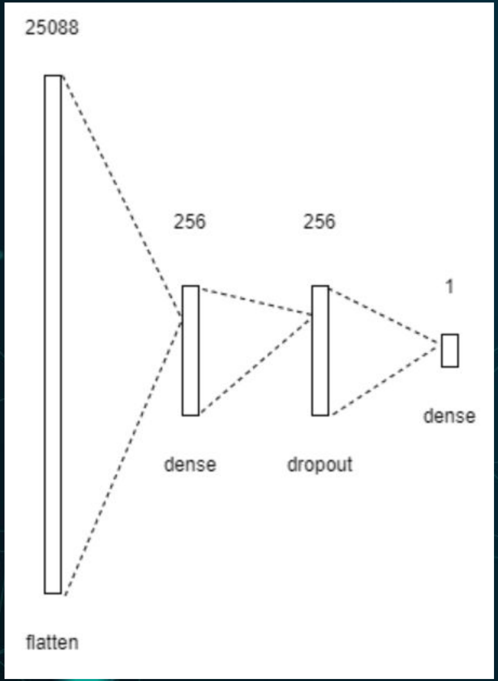




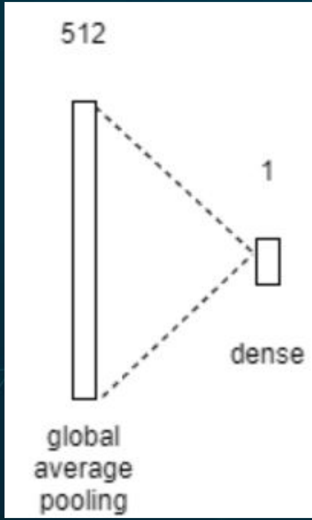
Convolutional Base - VGG16 Architecture



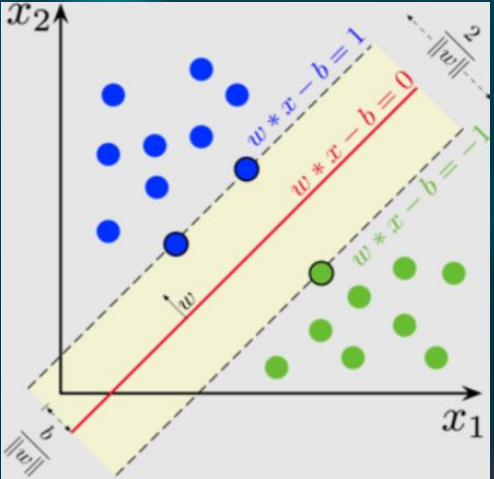
Classifier Architecture



Fully connected layers



Global average pooling



SVM

Performance Metrics

05

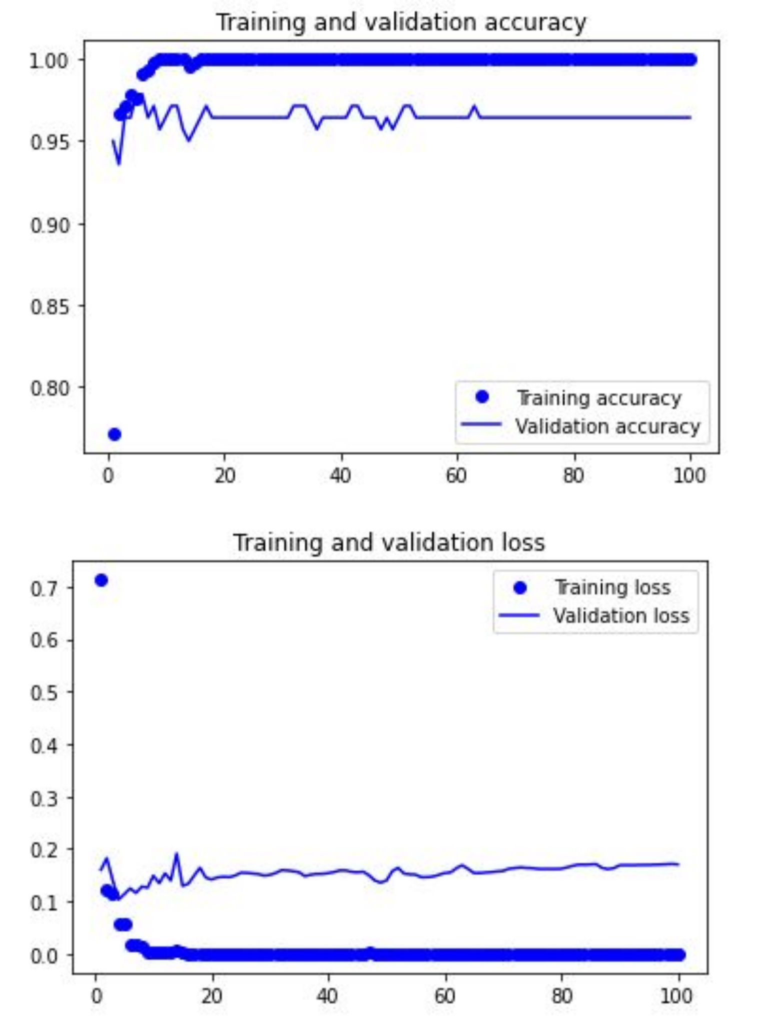
Metrics used to determine the performance of the ML Algorithm

Performance Metrics

“Numbers have an important story to tell. They rely on you to give them a voice.”

- Performance metrics are used to find the effectiveness of a model based on some metric using the test dataset
- Different performance metrics are used to evaluate different Machine Learning Algorithms
- The metrics that we choose to evaluate our machine learning model is very important
- Choice of metrics influences how the performance of machine learning algorithms is measured and compared

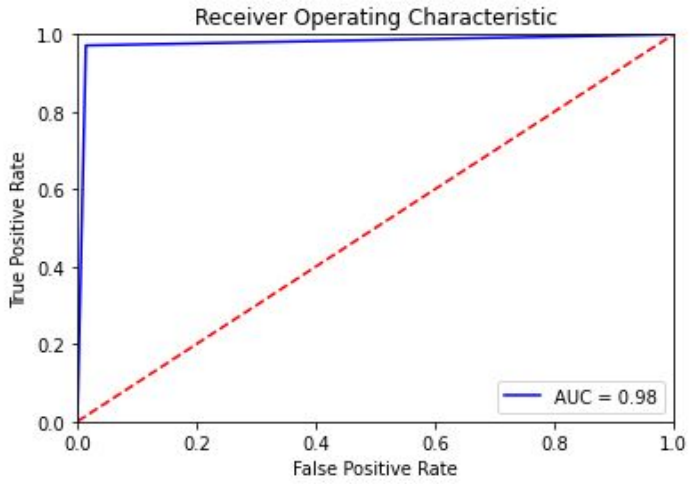
<table>
<tr><td colspan="2"></td><th colspan="2">True condition</th><td colspan="3"></td></tr>
<tr><td></td><td>Total population</td><td>Condition positive</td><td>Condition negative</td><td>Prevalence $= \frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$</td><td colspan="2">Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$</td></tr>
<tr><td rowspan="2">Predicted condition</td><td>Predicted condition positive</td><td>True positive</td><td>False positive,
Type I error</td><td>Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$</td><td colspan="2">False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$</td></tr>
<tr><td>Predicted condition negative</td><td>False negative,
Type II error</td><td>True negative</td><td>False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$</td><td colspan="2">Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$</td></tr>
<tr><td colspan="2" rowspan="2"></td><td>True positive rate (TPR), Recall, Sensitivity, probability of detection, Power $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$</td><td>False positive rate (FPR), Fall-out, probability of false alarm $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$</td><td>Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$</td><td rowspan="2">Diagnostic odds ratio (DOR) $= \frac{\text{LR+}}{\text{LR}-}$</td><td rowspan="2">$F_1$ score = $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$</td></tr>
<tr><td>False negative rate (FNR), Miss rate $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$</td><td>Specificity (SPC), Selectivity, True negative rate (TNR) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$</td><td>Negative likelihood ratio (LR−) $= \frac{\text{FNR}}{\text{TNR}}$</td></tr>
</table>



Output:

Fully-Connected Layers

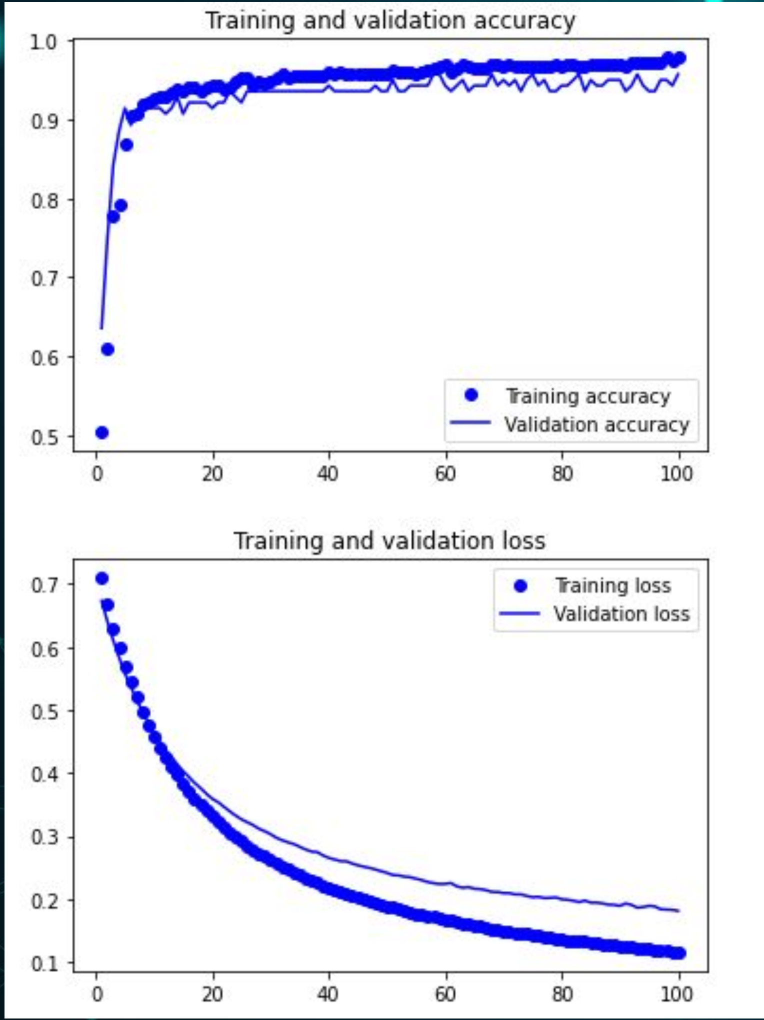
```
Confusion Matrix:
[[69  1]
 [ 2 68]]
Accuracy:  0.9785714285714285
Specificity:  0.9857142857142858
Precision/Positive Predictive Value:  0.9855072463768116
Negative Predictive Value:  0.971830985915493
Recall/Sensitivity:  0.9714285714285714
False Positive Rate:  0.3333333333333333
False Negative Rate:  0.02857142857142857
Positive Likelihood Ratio:  2.9142857142857146
Negative Likelihod Ratio:  0.028985507246376812
Diagnostic Odds Ratio:  100.54285714285716
False Omission Rate:  0.028169014084507043
F1 Score:  0.9784172661870504
Area under ROC curve:  0.9785714285714286
```



Output:

Performance Metrics

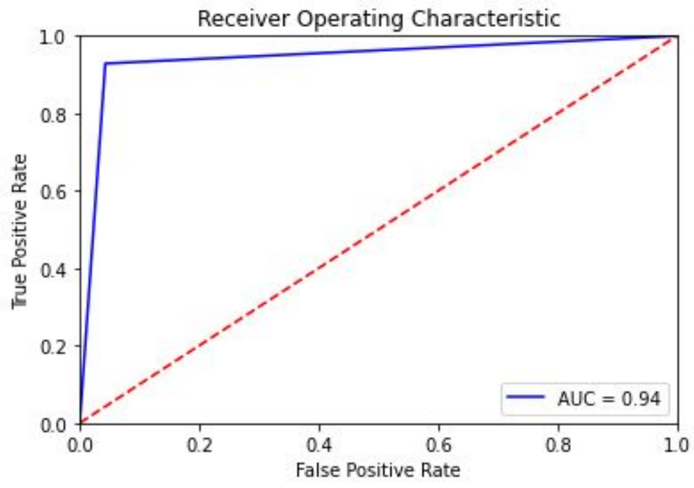
Fully-Connected Layers



Output:

Global Average Pooling

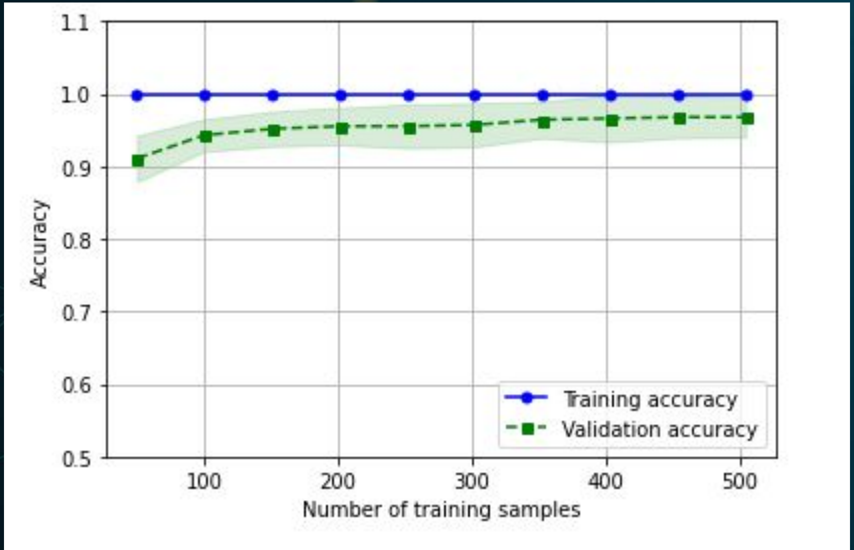
```
Confusion Matrix:
[[67  3]
 [ 5 65]]
Accuracy:  0.9428571428571428
Specificity:  0.9571428571428572
Precision/Positive Predictive Value:  0.9558823529411765
Negative Predictive Value:  0.9305555555555556
Recall/Sensitivity:  0.9285714285714286
False Positive Rate:  0.375
False Negative Rate:  0.07142857142857142
Positive Likelihood Ratio:  2.4761904761904763
Negative Likelihod Ratio:  0.07462686567164178
Diagnostic Odds Ratio:  33.180952380952384
False Omission Rate:  0.06944444444444445
F1 Score:  0.9420289855072465
Area under ROC curve:  0.9428571428571428
```



Output:

Performance Metrics

Global Average Pooling



Output:

Linear Support Vector Machines

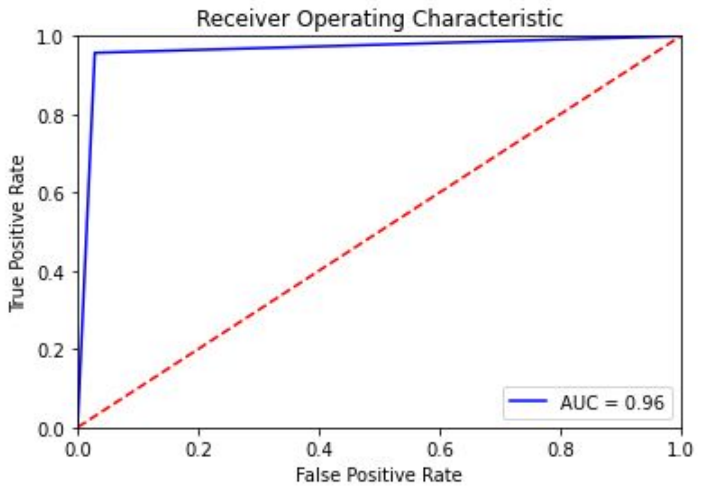
Learning Curve

Output:

Performance Metrics

Linear Support Vector Machines

```
Confusion Matrix:
[[68  2]
 [ 3 67]]
Accuracy:  0.9642857142857143
Specificity:  0.9714285714285714
Precision/Positive Predictive Value:  0.9710144927536232
Negative Predictive Value:  0.9577464788732394
Recall/Sensitivity:  0.9571428571428572
False Positive Rate:  0.4
False Negative Rate:  0.04285714285714286
Positive Likelihood Ratio:  2.392857142857143
Negative Likelihod Ratio:  0.04411764705882353
Diagnostic Odds Ratio:  54.238095238095234
False Omission Rate:  0.04225352112676056
F1 Score:  0.9640287769784173
Area under ROC curve:  0.9642857142857144
```



Comparison of the Accuracy of the Three Different Classifiers

	Fully Connected Layer	Global average Pooling	SVM
Accuracy	97.86%	94.29%	96.43%

Summary 06

Summary of the core approaches in the presentation

Summary

- Presented the concepts of transfer learning, convolutional neural networks, and pre-trained models.
- Defined the basic fine-tuning strategies to repurpose a pre-trained model.
- Described a structured approach to decide which fine-tuning strategy should be used, based on the size and similarity of the dataset.
- Introduced three different classifiers that can be used on top of the features extracted from the convolutional base.
- Provided a insights about each of the three classifier.
- Analysed each of the three classifiers.
- Drew ROC curve for each of the three classifiers.
- Reported the accuracy of the three classifiers.

Thank You