UCS1603 Introduction to Machine Learning

ML PROJECT REPORT

POTHOLE & PLAIN ROAD CLASSIFICATION USING TRANSFER LEARNING & CNN

TEAM MEMBERS

Lakshmi Priya B - 185001083

Nivedhitha D - 185001104

Raghav R - 185001119

Sakthi Sairaj - 185001134

INDEX

Abstract	**3**
Project Importance and Necessity	**4**
Problem Statement	**4**
Dataset Details	**5**
Dataset Stats	**5**
Directory Structure	**5**
Methodology	**6**
Transfer Learning	6
Convolutional Neural Networks	6
CNN Architecture	7
Repurposing a pre-trained model	7
Transfer Learning Process	8
Classifiers on top of deep CNNs	9
Fully-connected Layers	9
Global Average Pooling	10
Linear Support Vector Machines	11
Proposed Workflow	12
PREPARE DATA	12
EXTRACT FEATURES	12
BUILD CLASSIFIERS	12
Fully-connected layers	12
Global average pooling	12
Linear support vector machines	12
TRAIN MODEL	12
PERFORMANCE ANALYSIS	12
Model Architecture	**13**
Convolutional Base - VGG16 Architecture	14
Classifier Architecture	14
Performance Metrics	**15**
Performance Analysis	16
Fully-Connected Layers	16
Global Average Pooling	17
Linear Support Vector Machines	18
Comparison of the Accuracy of the Three Different Classifiers	18
Conclusion	**19**

Abstract

In this project, we try to implement a transfer learning and CNN based solution to solve image classification problems using data from the Pothole and Plain Road Images Kaggle dataset is used in this project. The goal of this project is to build models that identify if the image has potholes or is a plain road without potholes. Accordingly, it is a binary classification problem.

In this project, we want to use a solution based on pre-trained models. We will use models that are composed of two parts:

- Convolutional base
- Classifier

Our approach will use the convolutional base to extract features, using them to train a classifier to classify the input image as a pothole or plain road. Therefore, the features extracted from the convolutional base will be the same for all classifiers studied in this example.

Solution using *3 MODELS*

1. Fully connected layer classifier
2. Global Average Pooling classifier
3. Linear Support Vector Machine classifier

Keywords: machine learning, deep-learning, svm, scikit-learn, cnn, performance metrics, convolutional neural networks, transfer learning, vgg16, svm model, keras, svm classifier, fully connected network, linear svm, global average pooling, cnn classification, detection model, plotting performance graphs, pothole detection

Project Importance and Necessity

*Every year around **3,597** people die due to potholes. More than 30% of people die due to potholes. The Ministry of Road Transport and Highways provided figures that over 9300 deaths, **25000** injured in the last three years due to potholes and more than **25,000** people are getting injured due to potholes. [source]*

So, it is extremely vital to detect potholes and fix them as soon as possible to avoid the loss of valuable lives. We have built a model for Pothole Detection using Transfer Learning and CNN which has very high accuracy. Our model can be deployed very easily and even the images from CCTV cameras alongside roads can be fed as input to our model. Once the potholes are detected from CCTV camera images or images acquired from any other means, a road safety team can be dispatched to fix the pothole on the road.

Problem Statement

Using deep learning and transfer learning techniques to solve the binary image classification problem of pothole detection by adopting an accurate model for savings in training time and computational efficiency.

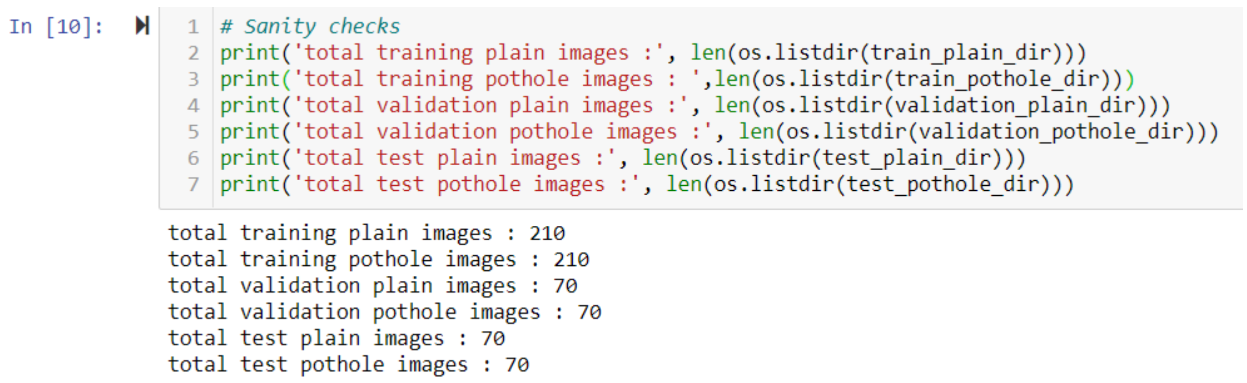
Pothole

Plain road

Dataset Details

NAME	Pothole and Plain road images
SOURCE	Kaggle
RECORD TYPE	.jpg, .jpeg, .png, .gif images
CLASSES	Binary classification - pothole and plain road
NUMBER OF RECORDS	700 (350 in each class)
TRAINING:VALIDATION:TEST SPLIT	60:20:20 ratio (210-70-70 images)
FEATURE EXTRACTION	Convolutional Base (Deep Learning)

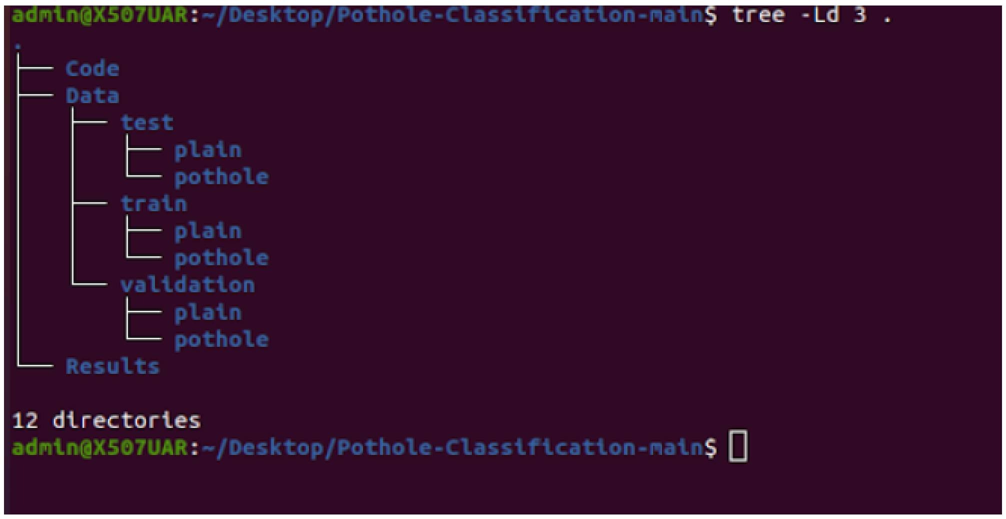
Dataset Stats



```
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

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).

Convolutional Neural Networks

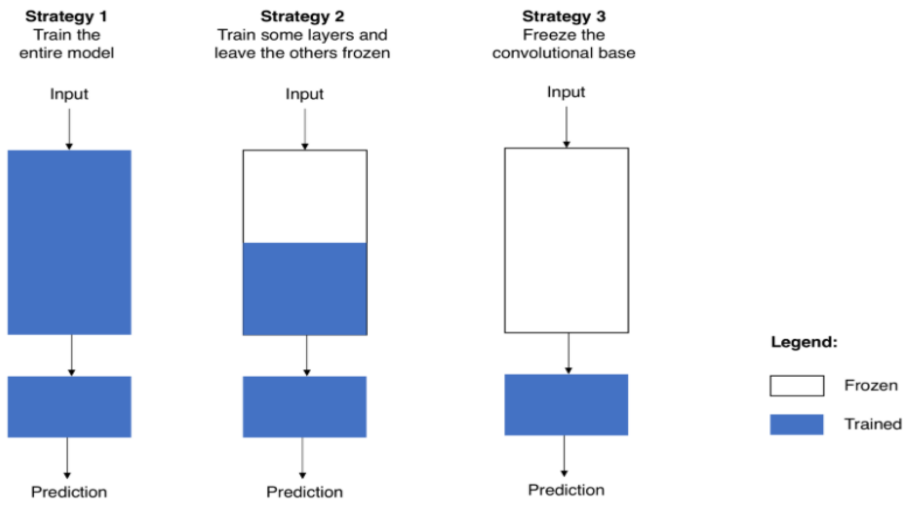
- Several pre-trained models that are used in transfer learning are based on large CNNs.
- High performance and ease of training are two of the main factors driving the popularity of CNN over the last years.
- A typical CNN has two parts:
    - Convolutional base, which is composed by a stack of convolutional 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.

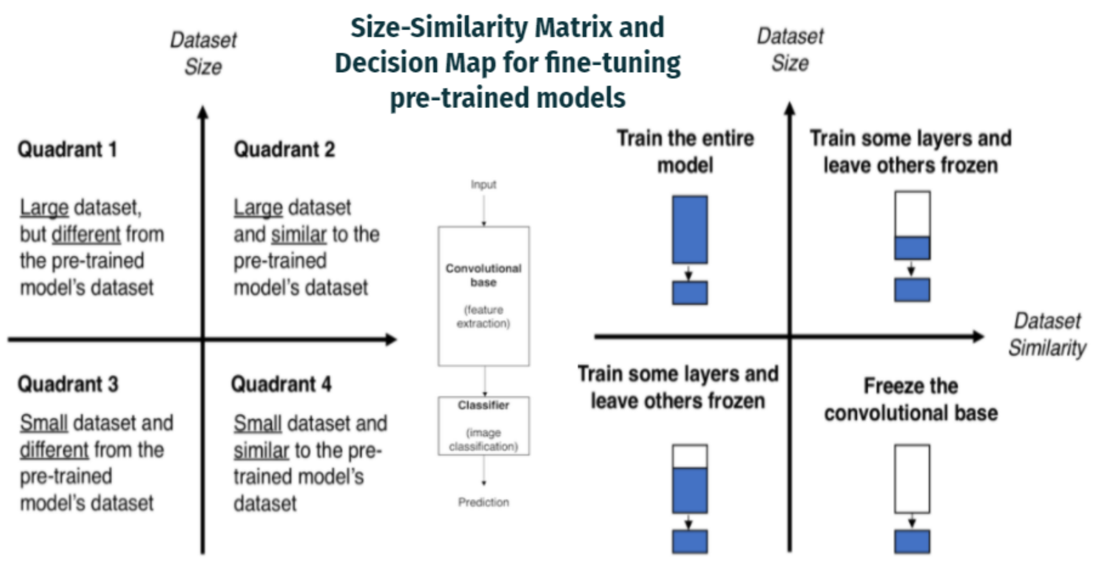
Repurposing a pre-trained model

- 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 and 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



Transfer Learning Process

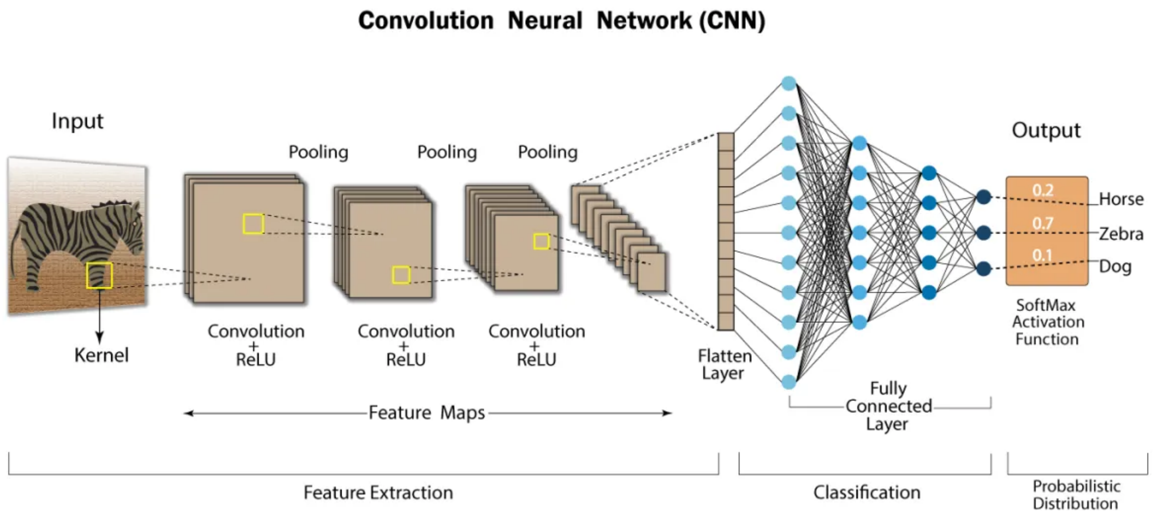
1. Select a pre-trained model based on the problem statement
2. Classify your problem according to the Size-Similarity Matrix
    a. Classifies your computer vision problem considering the size of your dataset and its similarity to the dataset in which your pre-trained model was trained
    b. As a rule 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 Similarity
Quadrant 1
Large dataset, but different from the pre-trained model's dataset
Quadrant 2
Large dataset and similar to the pre-trained model's dataset
Quadrant 3
Small dataset and different from the pre-trained model's dataset
Quadrant 4
Small dataset and similar to the pre-trained model's dataset
Input
Convolutional base
(feature extraction)
Classifier
(image classification)
Prediction
Train the entire model
Train some layers and leave others frozen
Train some layers and leave others frozen
Freeze the convolutional base

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:
    - Convolutional base, which performs feature extraction
    - Classifier, which classifies the input image based on the features extracted by the convolutional base

Fully-connected Layers



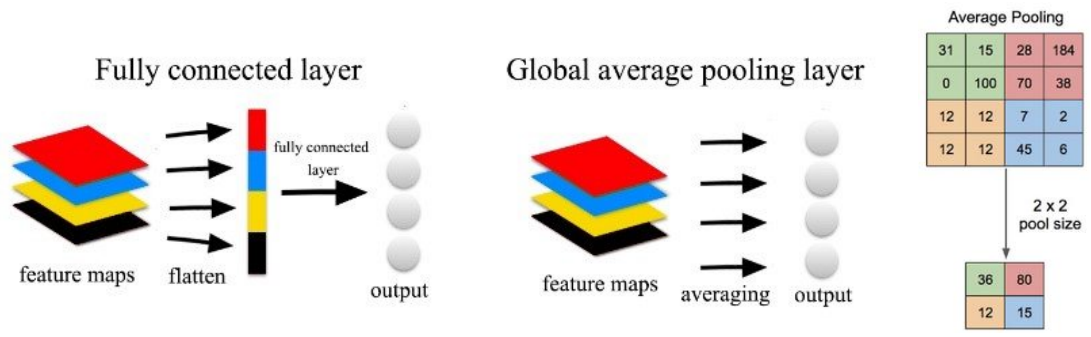
- For image classification problems, the standard approach is to use a stack of fully-connected layers followed by a softmax activated layer.
- The softmax layer outputs the probability distribution over each possible class label and then we just need to classify the image according to the most probable class.
- In comparison to the other layers, this category of layers has the highest number of parameters because every neuron is connected to every other neuron. The number of parameters is the product of the number of neurons in the current layer **c** and the number of neurons on the previous layer **p** plus the bias term. Thus number of parameters here are: **((current layer neurons c * previous layer neurons p)+1*c)**.

- **Softmax Function:**

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \dots, K \text{ and } \mathbf{z} = (z_1, \dots, z_K) \in \mathbb{R}^K$$

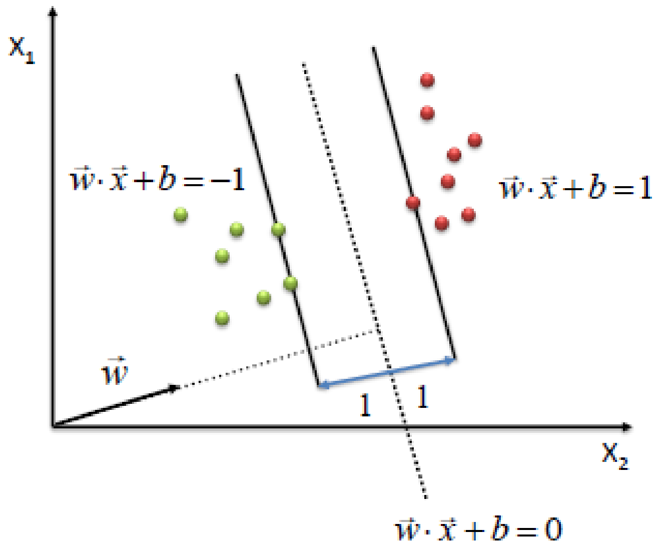
The softmax function takes as input a vector *z* of *K* real numbers, and normalizes it into a probability distribution consisting of *K* probabilities proportional to the exponentials of the input numbers.

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.
- The activation maps, called feature maps, capture the result of applying the filters to input, such as the input image or another feature map. The idea of visualizing a feature map for a specific input image would be to understand what features of the input are detected or preserved in the feature maps.
- On two-dimensional feature maps, pooling is typically applied in 2×2 patches of the feature map with a stride of (2,2).
- Average pooling involves calculating the average for each patch of the feature map. This means that each 2×2 square of the feature map is down sampled to the average value in the square.

Linear Support Vector Machines


X_1
$\vec{w}\cdot\vec{x}+b=-1$
$\vec{w}\cdot\vec{x}+b=1$
$\vec{w}$
1
1
X_2
$\vec{w}\cdot\vec{x}+b=0$

$$\boxed{\max \frac{2}{\|w\|}}$$

$$\begin{aligned}&s.t.\\ &(w\cdot x+b)\geq 1, \forall x \text{ of class 1}\\ &(w\cdot x+b)\leq -1, \forall x \text{ of class 2}\end{aligned}$$

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

In the case of support-vector machines, a data point is viewed as a p-dimensional vector (a list of p numbers), and we want to know whether we can separate such points with a $(p-1)$ - dimensional hyperplane.

This is called a linear classifier. There are many hyperplanes that might classify the data. One reasonable choice as the best hyperplane is the one that represents the largest separation, or margin, between the two classes. So we choose the hyperplane so that the distance from it to the nearest data point on each side is maximized. If such a hyperplane exists, it is known as the *maximum-margin hyperplane* and the linear classifier it defines is known as a *maximum-margin classifier*.

Proposed Workflow

PREPARE DATA

Choose an 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.

BUILD CLASSIFIERS

Fully-connected layers

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

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.

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).

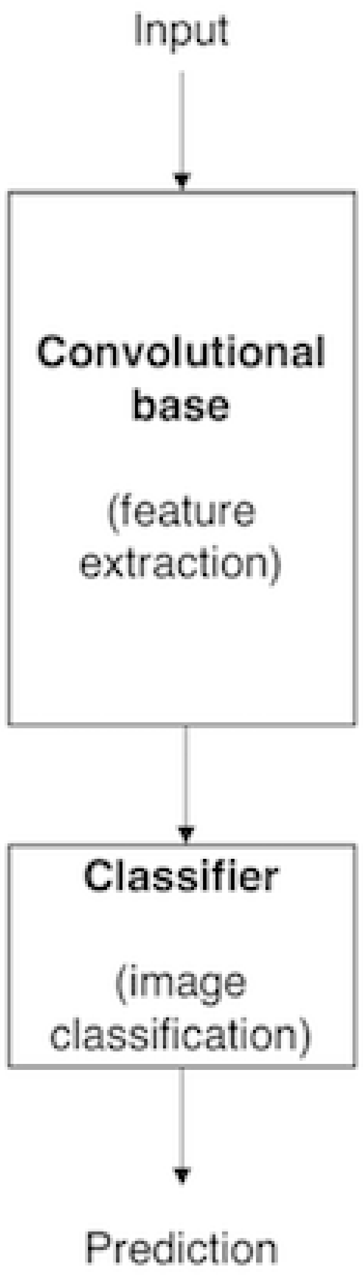
TRAIN MODEL

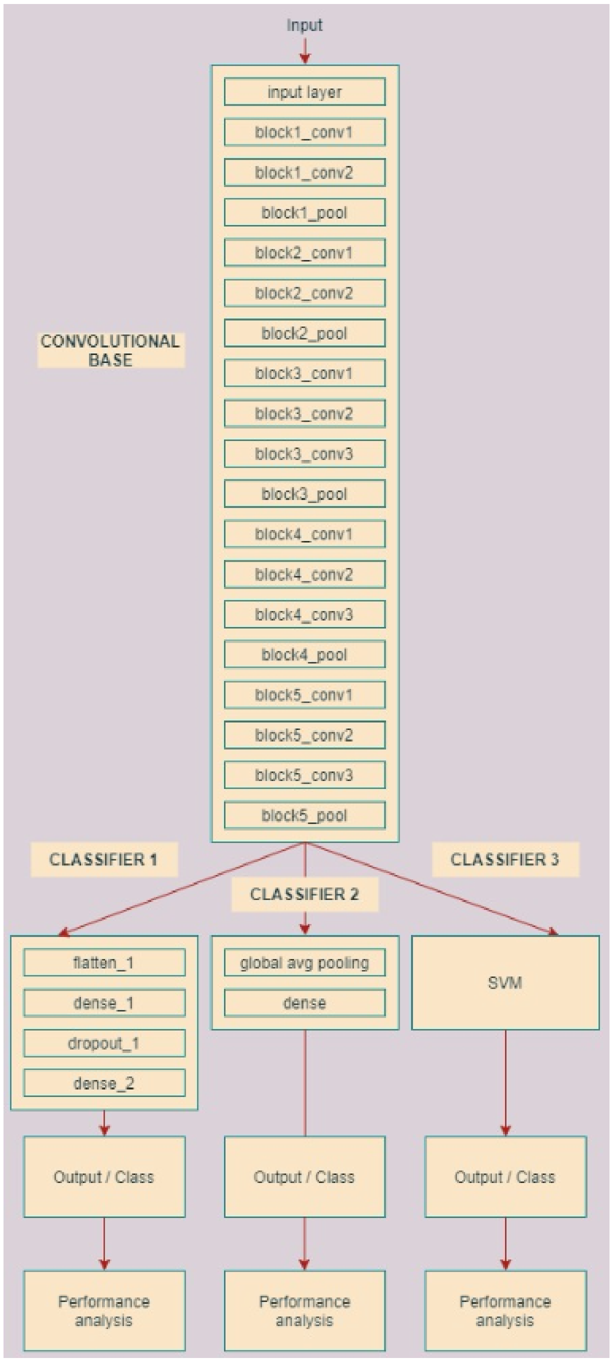
Train the model by fixing the epochs and batch size.

PERFORMANCE ANALYSIS

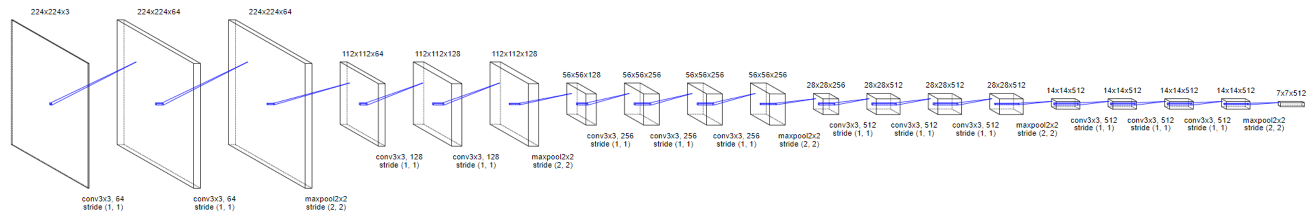
Analyse the performance of the model using various performance metrics.

Model Architecture

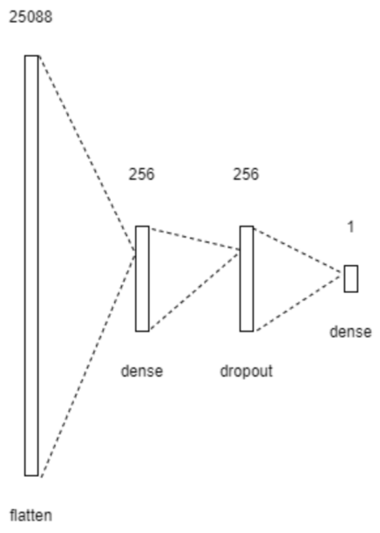




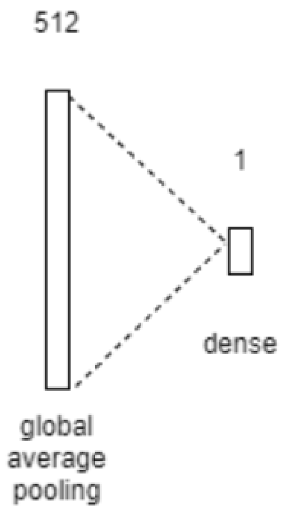
Convolutional Base - VGG16 Architecture


224x224x3
224x224x64
224x224x64
112x112x64
112x112x128
112x112x128
56x56x128
56x56x256
56x56x256
56x56x256
28x28x256
28x28x512
28x28x512
28x28x512
14x14x512
14x14x512
14x14x512
14x14x512
7x7x512
conv3x3, 64 stride (1, 1)
conv3x3, 64 stride (1, 1)
maxpool2x2 stride (2, 2)
conv3x3, 128 stride (1, 1)
conv3x3, 128 stride (1, 1)
maxpool2x2 stride (2, 2)
conv3x3, 256 stride (1, 1)
conv3x3, 256 stride (1, 1)
conv3x3, 256 stride (1, 1)
maxpool2x2 stride (2, 2)
conv3x3, 512 stride (1, 1)
conv3x3, 512 stride (1, 1)
conv3x3, 512 stride (1, 1)
maxpool2x2 stride (2, 2)
conv3x3, 512 stride (1, 1)
conv3x3, 512 stride (1, 1)
conv3x3, 512 stride (1, 1)
maxpool2x2 stride (2, 2)

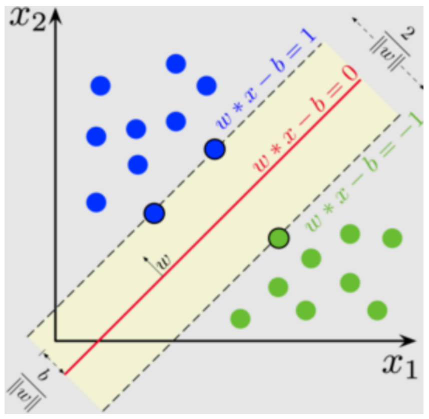
Classifier Architecture



Fully connected layers



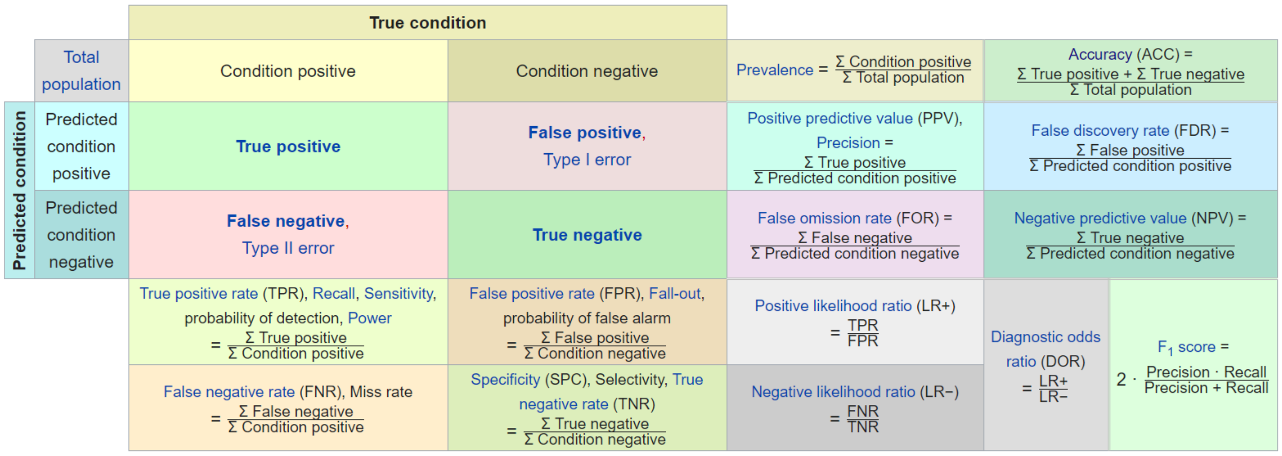
Global average pooling



SVM

Performance Metrics

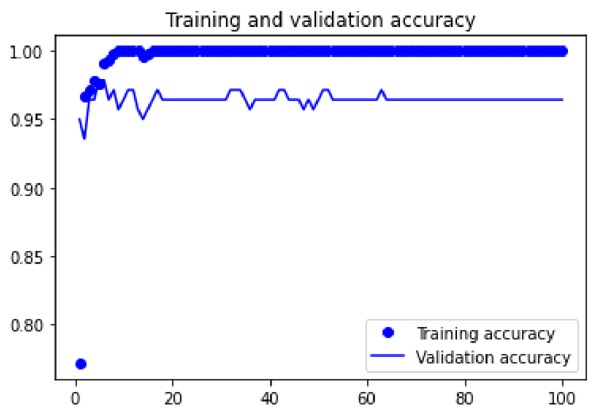
- “Numbers have an important story to tell. They rely on you to give them a voice.”
- Performance metrics are used to find the effectiveness of a model based on some metric using the test dataset.
- Different performance metrics are used to evaluate different Machine Learning Algorithms.
- The metrics 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.

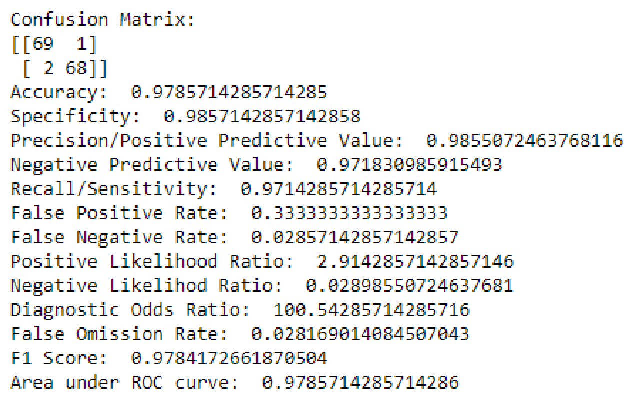


		True condition				
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	
Predicted condition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

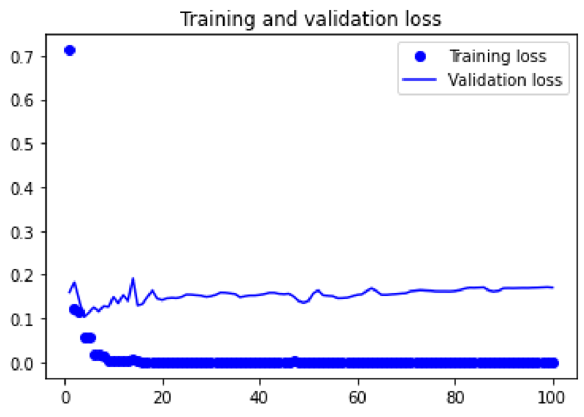
Performance Analysis

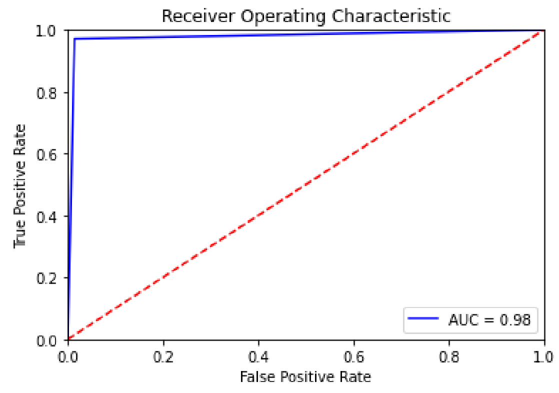
Fully-Connected Layers



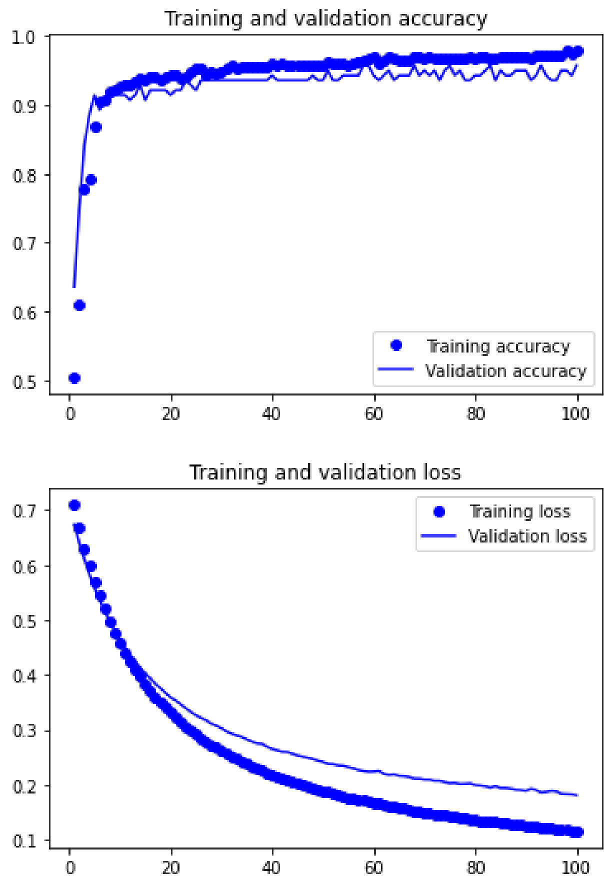


```
Confusion Matrix:
[[69  1]
 [ 2 68]]
Accuracy:  0.9785714285714285
Specificity:  0.9857142857142858
Precision/Positive Predictive Value:  0.9855072463768116
Negative Predictive Value:  0.971830985915493
Recall/Sensitivity:  0.9714285714285714
False Positive Rate:  0.3333333333333333
False Negative Rate:  0.02857142857142857
Positive Likelihood Ratio:  2.9142857142857146
Negative Likelihood Ratio:  0.028985507246376 81
Diagnostic Odds Ratio:  100.54285714285716
False Omission Rate:  0.028169014084507043
F1 Score:  0.9784172661870504
Area under ROC curve:  0.9785714285714286
```

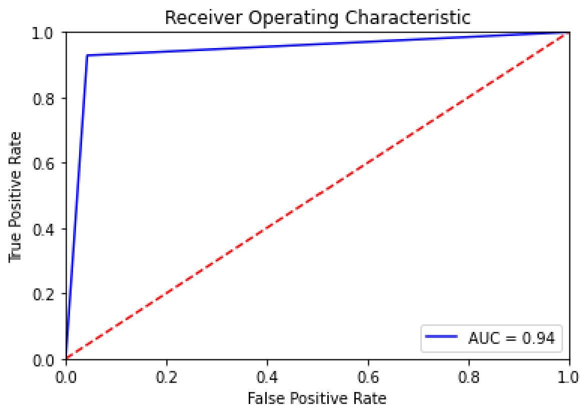




Global Average Pooling

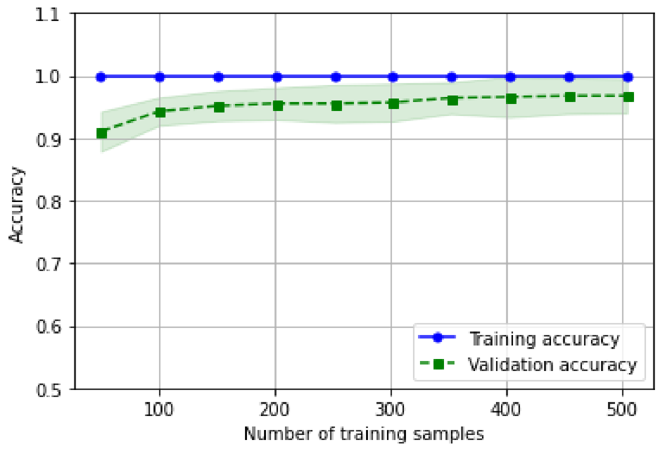


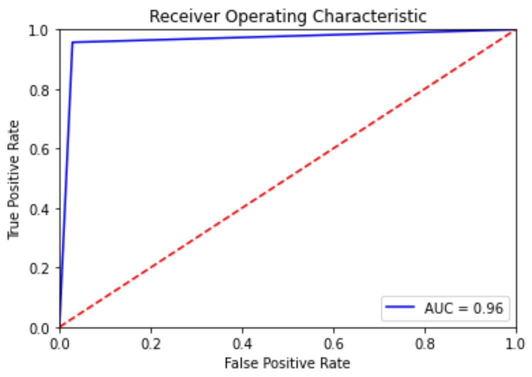
```
Confusion Matrix:
[[67  3]
 [ 5 65]]
Accuracy:  0.9428571428571428
Specificity:  0.9571428571428572
Precision/Positive Predictive Value:  0.9558823529411765
Negative Predictive Value:  0.9305555555555556
Recall/Sensitivity:  0.9285714285714286
False Positive Rate:  0.375
False Negative Rate:  0.07142857142857142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



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 Classifier	Global Average Pooling Classifier	Linear Support Vector Machine
Accuracy	97.86%	94.29%	96.43%

Conclusion

Thus, in this project we have

- 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 insights about each of the three classifiers.
- Analysed each of the three classifiers.
- Drew ROC curve for each of the three classifiers.
- Reported the accuracy of the three classifiers.