edureka!
a Veranda Enterprise | ILLINOIS TECH

POST GRADUATE PROGRAM IN GENERATIVE AI AND ML

Deep Learning and Neural Network Architectures

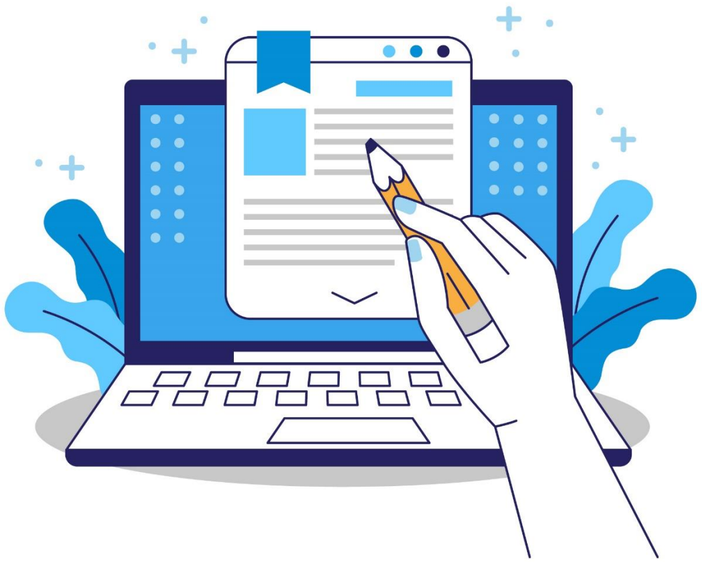
edureka!
a Veranda Enterprise

ILLINOIS TECH

Convolutional Neural Networks - II

Topics

- AlexNet and VGGNet architecture overview
- Transfer learning concepts
- Fine-tuning vs. feature extraction
- Freezing and unfreezing layers
- Using pre-trained models in Keras
- Confusion matrix and classification report
- Top-k accuracy and ROC curves
- Training CNNs on small datasets
- Visualizing class activation maps (CAMs)



Learning Objectives

By the end of this lesson, you will be able to:

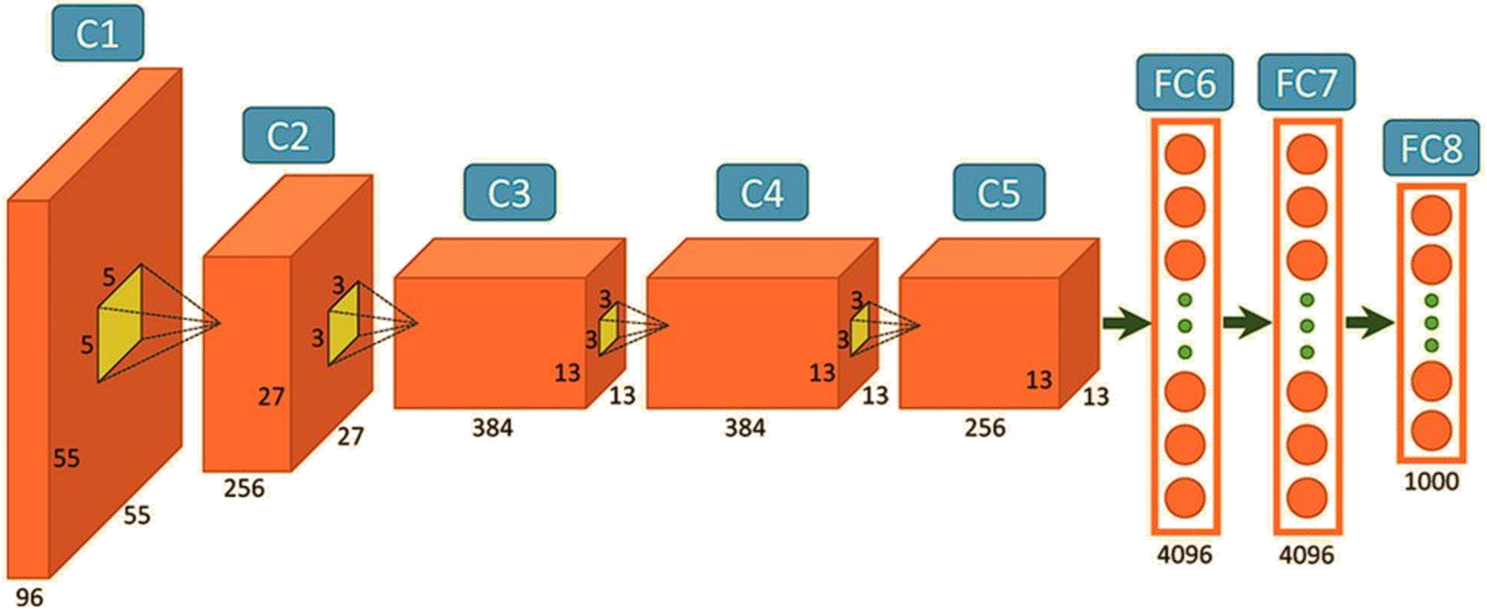
- Explore AlexNet and VGGNet architectures.
- Apply transfer learning in CNNs.
- Compare fine-tuning vs. feature extraction.
- Freeze and unfreeze CNN layers.
- Use pre-trained models with Keras.
- Evaluate with confusion matrix and reports.
- Measure performance with Top-k and ROC.
- Train CNNs on small datasets.
- Visualize attention with CAMs.

AlexNet and VGGNet Architecture Overview

AlexNet and its Architecture

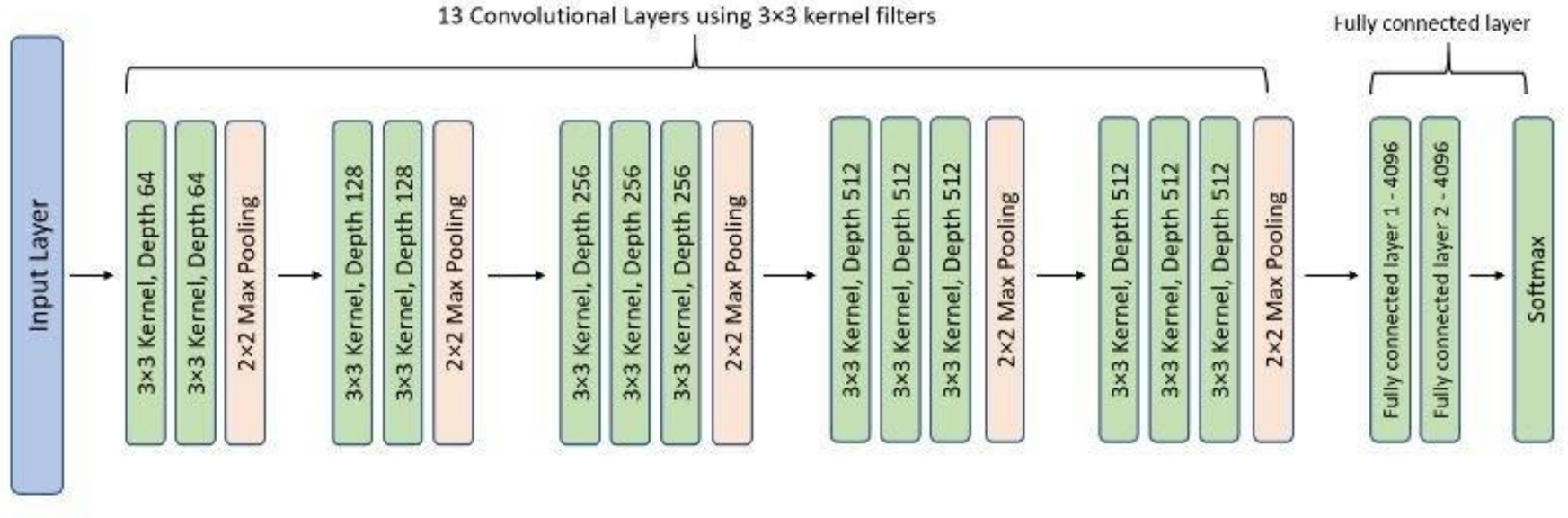
AlexNet, introduced in 2012, marked a paradigm shift in computer vision by achieving unprecedented performance in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC).

AlexNet comprises **eight layers**: five convolutional layers followed by three fully connected layers.



VGGNet and its Architecture

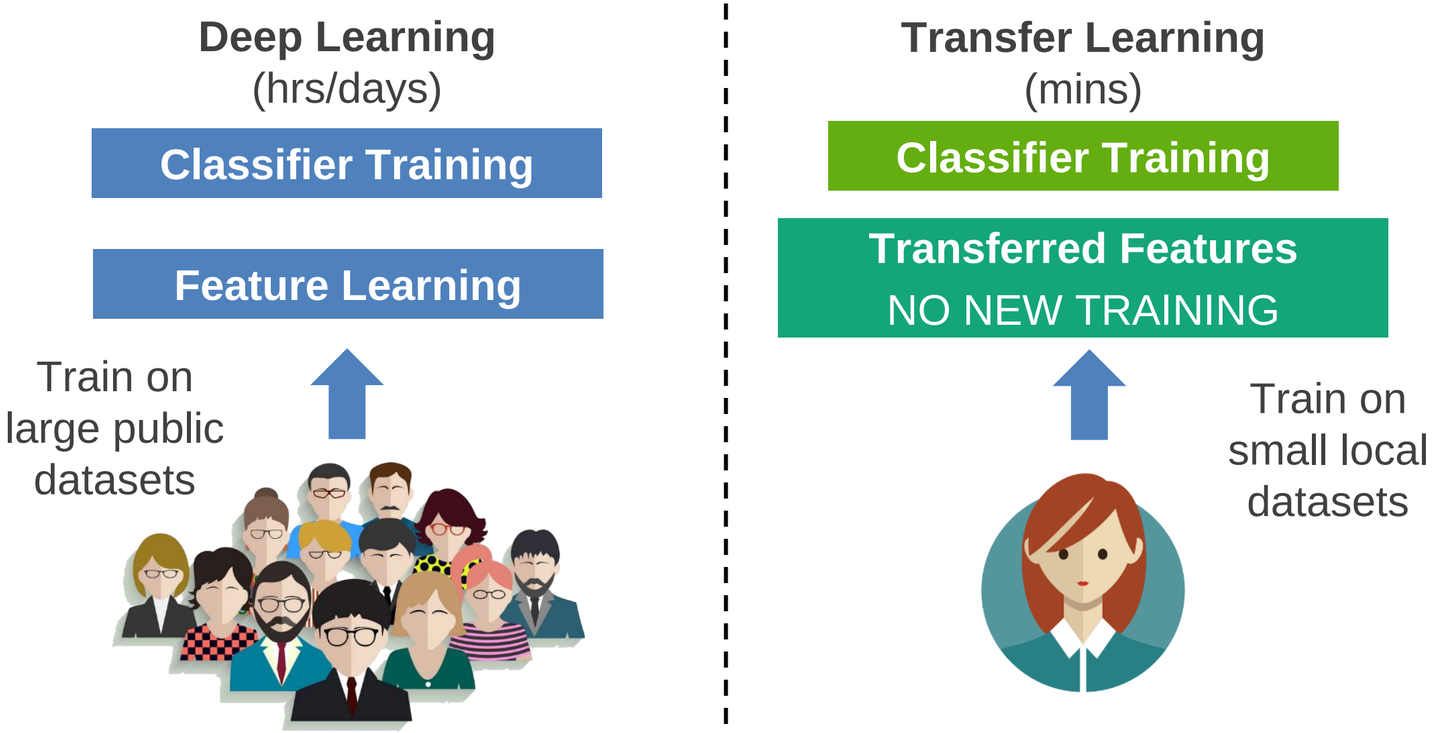
Developed by the Visual Geometry Group at Oxford, VGGNet (2014) demonstrated that increasing network depth with small convolutional filters significantly improves recognition accuracy.



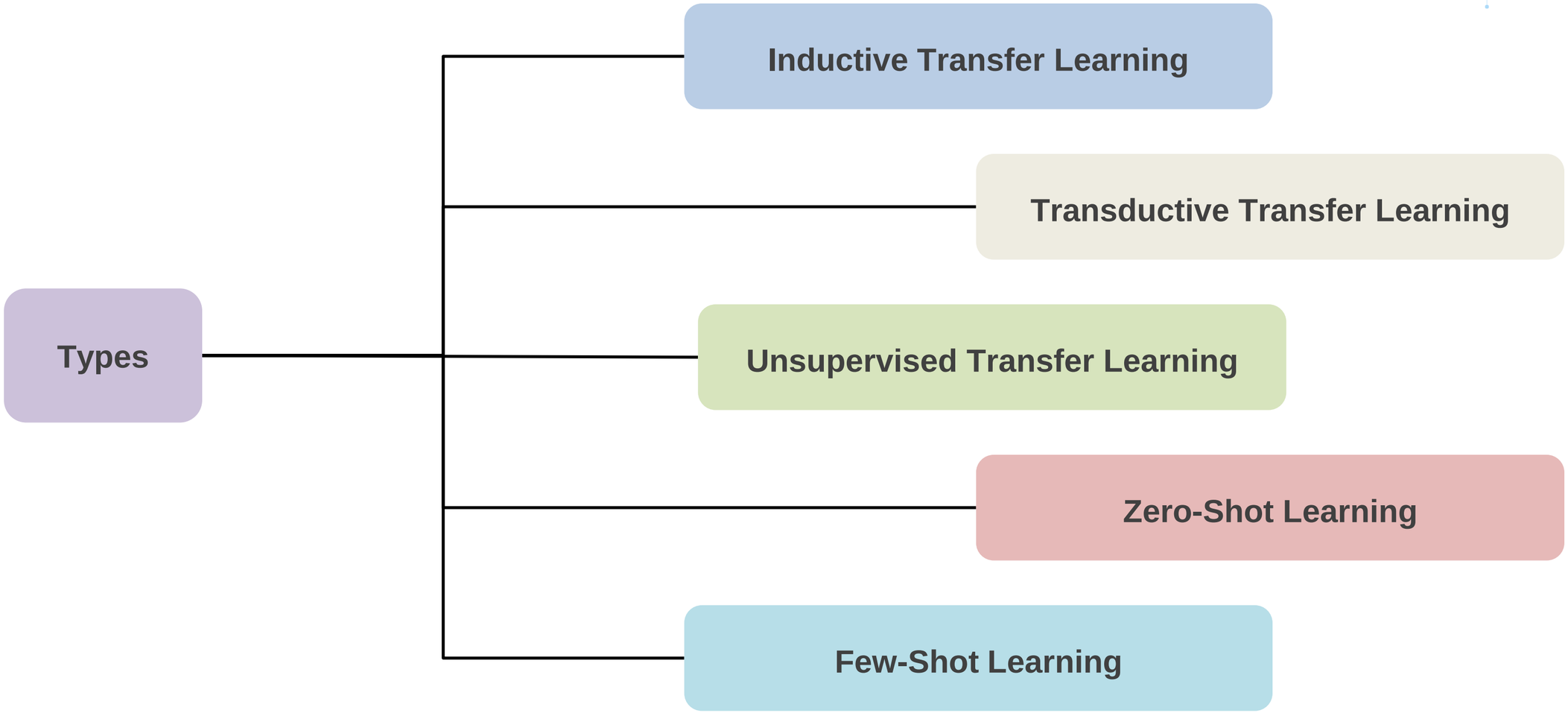
Transfer Learning Concepts

Transfer Learning

Transfer Learning is one of the mighty tasks of deep learning that allows us to take the knowledge the neural network has learned from one task(A) and apply that knowledge in another task (B).



Types of Transfer Learning



Fine-Tuning vs. Feature Extraction

Fine-Tuning vs. Feature Extraction

These are two foundational strategies in transfer learning with CNNs. Both leverage **pre-trained models** but differ in how much of the **original model is updated for a new task**.

Fine Tuning

Involves unfreezing some (often the deeper) layers of a pre-trained CNN and retraining them, along with the new classifier layers, on the new dataset

Feature Extraction

Uses a pre-trained CNN as a fixed feature extractor. The convolutional base (all or most layers) is frozen-its weights are not updated during training on the new task

Comparison Table

Aspect	Feature Extraction	Fine Tuning
Layers Trained	Only new classifier layers	New classifier + some pre-trained layers
Model Adaptability	Limited	High
Data Requirement	Low	Moderate to High
Computational Cost	Low	Higher
Overfitting Risk	Low	Higher (if data is insufficient)
Use Case	Small, similar datasets	Larger, more distinct datasets

Freezing and Unfreezing Layers

What is Freezing and Unfreezing in CNN?

Unfreezing layers

Allows selected (usually deeper) layers to be fine-tuned (i.e., trained further on new data).

Freezing layers

Prevents certain layers from updating their weights during training when we use a pre-trained model.

Why Freeze?

Keep learned features (e.g., edges, textures) from a large dataset like ImageNet.

Why Unfreeze?

Adapt deeper layers to the new dataset (helps in domain-specific tuning).

Freezing and Unfreezing

Example Code: Freezing All Layers

```
# Load a pre-trained model
from tensorflow.keras.applications import VGG16
base_model = VGG16(weights='imagenet', include_top=False)

# Freeze all layers
for layer in base_model.layers:
    layer.trainable = False
```

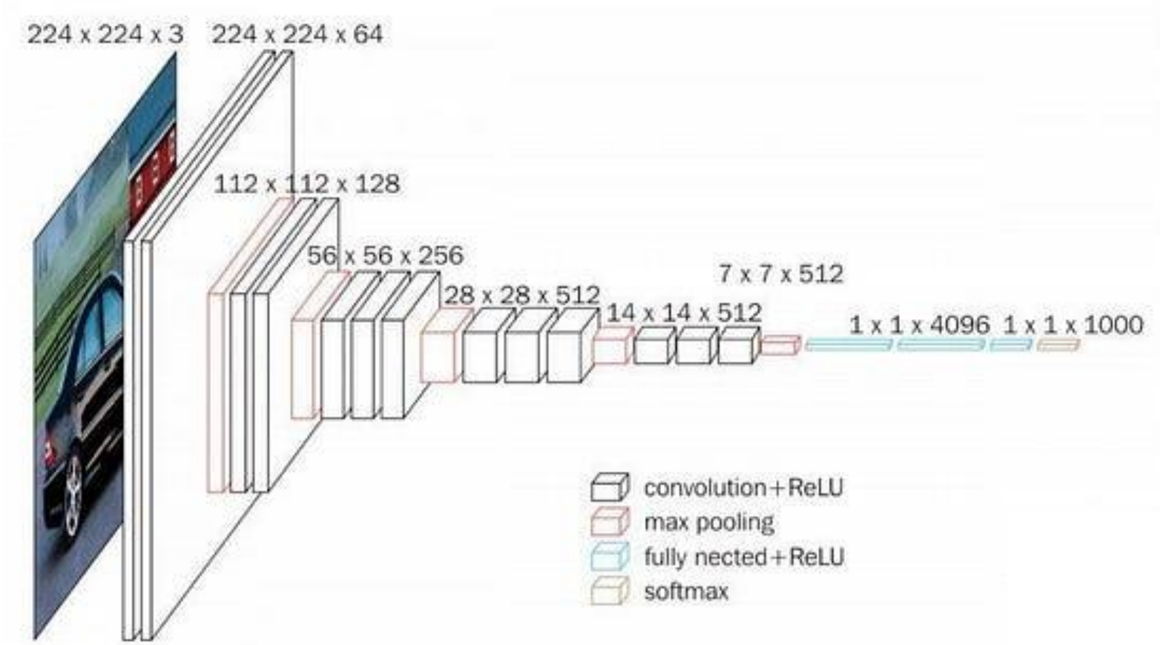
Example Code: Unfreezing Specific Layers

```
# Unfreeze last 4 layers
for layer in base_model.layers[-4:]:
    layer.trainable = True
```

Using Pre-Trained Models in Keras

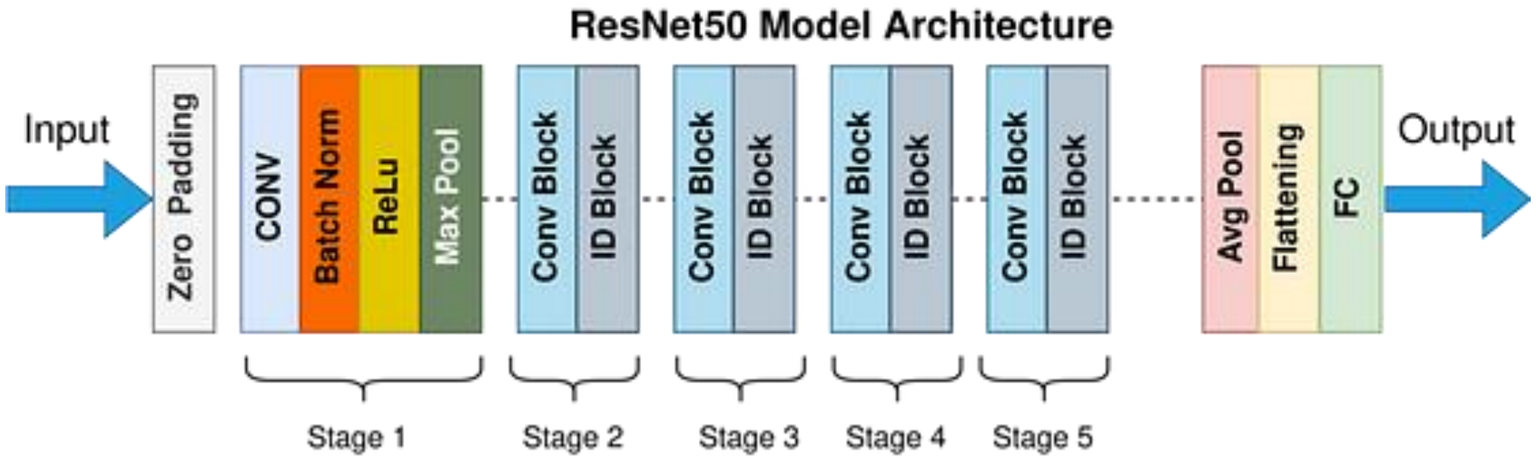
Model 1: VGG16

VGG16 is a popular convolutional neural network model trained on ImageNet. It has 16 layers and is often used for image classification tasks or as a feature extractor.



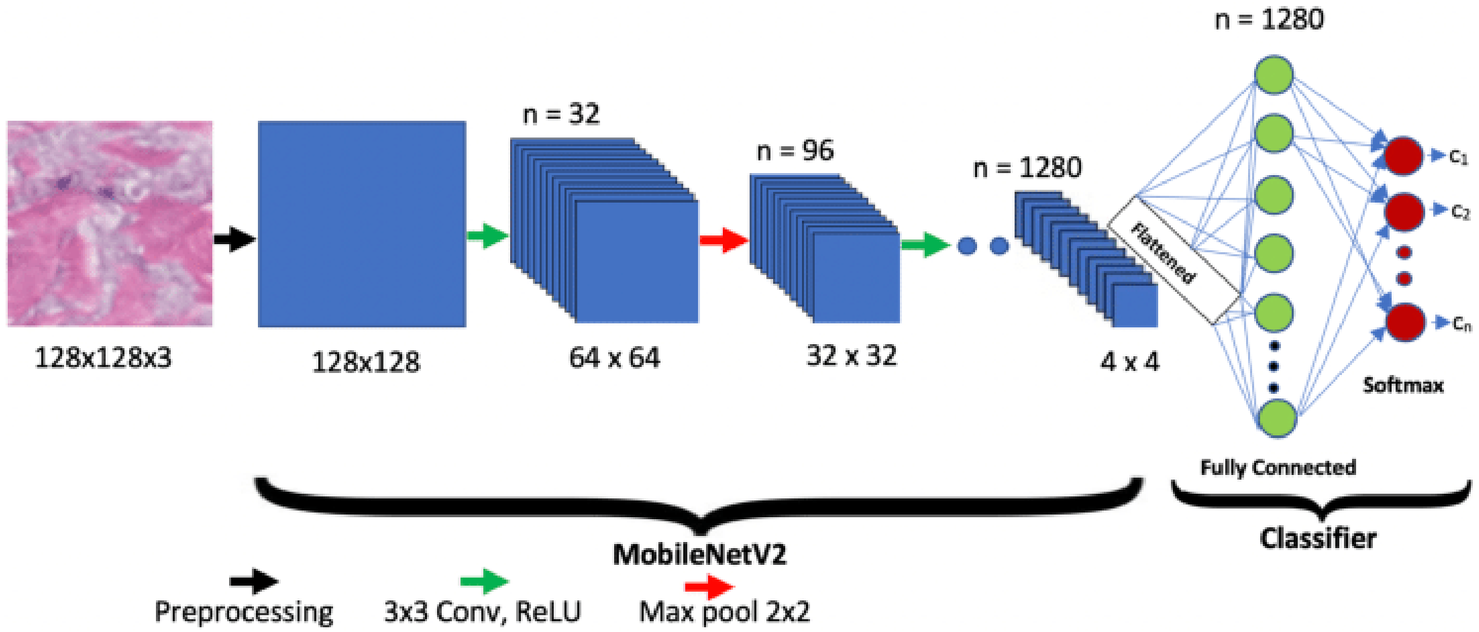
Model 2: ResNet50

ResNet50 (Residual Network) is a 50-layer deep CNN that uses skip connections to handle vanishing gradients and train very deep networks effectively.



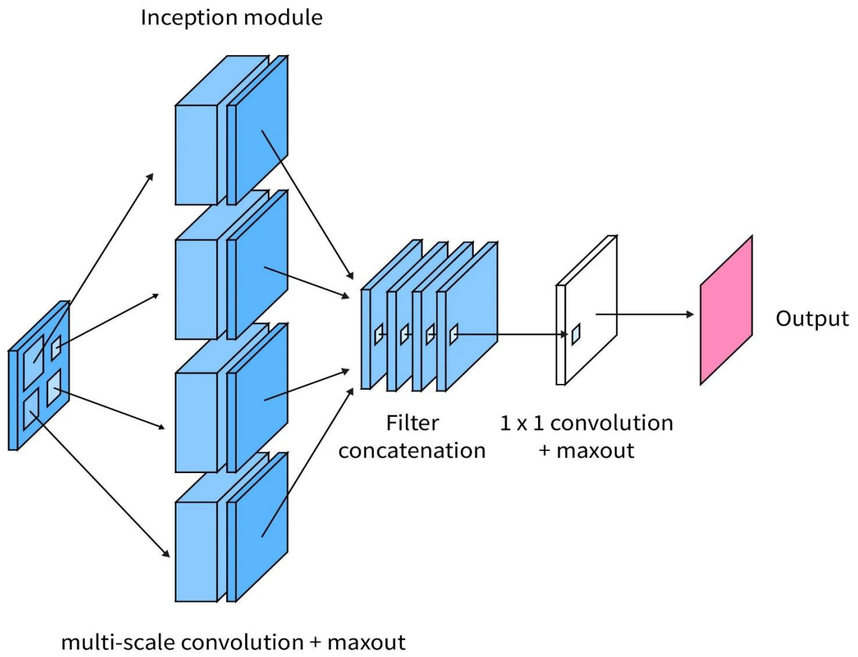
Model 3: MobileNetV2

MobileNetV2 is a lightweight CNN optimized for mobile and edge devices. It's efficient and fast, making it ideal for real-time applications.



Model 4: InceptionNet

InceptionNet (also called **GoogLeNet**) is a deep convolutional neural network that uses Inception modules, which apply multiple filter sizes (1x1, 3x3, 5x5) in parallel, allowing the model to learn both fine and coarse features efficiently



Confusion Matrix and Classification Report

What Is a Confusion Matrix?

A confusion matrix is a tabular summary of a **classifier's performance**, showing the counts of correct and incorrect predictions broken down by each actual and predicted class.

- **True Positives:** correctly labeled positives.
- **False Negatives:** positives mislabeled as negatives (missed).
- **False Positives:** negatives mislabeled as positives (noise).
- **True Negatives:** correctly labeled negatives.

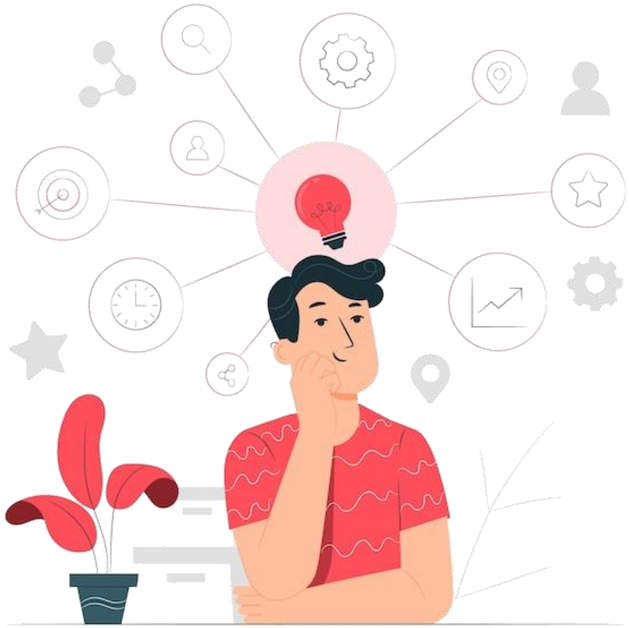
	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

Why Confusion Matrix?

Deep learning classifiers on text don’t exist in a vacuum—their performance often hinges on how well you’ve preprocessed your data.

By examining a confusion matrix after an initial model run, you can:

- **pinpoint specific kinds of errors** (e.g., false positives vs. false negatives) and
- tailor your **text-cleaning and feature-engineering** steps to reduce them.



Classification Report

What is a Classification Report?

- A classification report gives key metrics:
- Precision: How many predicted positives are truly positive?
- Recall: How many actual positives were correctly predicted?
- F1-Score: Harmonic mean of precision and recall
- Support: Number of actual samples per class

	precision	recall	f1-score	support
airplane	0.63	0.37	0.47	1000
automobile	0.75	0.30	0.43	1000
bird	0.35	0.20	0.25	1000
cat	0.31	0.37	0.33	1000
deer	0.40	0.30	0.34	1000
dog	0.47	0.23	0.31	1000
frog	0.46	0.51	0.48	1000
horse	0.38	0.65	0.48	1000
ship	0.48	0.66	0.55	1000
truck	0.40	0.72	0.51	1000
accuracy			0.43	10000
macro avg	0.46	0.43	0.42	10000
weighted avg	0.46	0.43	0.42	10000

Top-K Accuracy and ROC Curves

Top-K Accuracy in CNN

Top-k accuracy measures the proportion of samples for which the correct label is among the model's top k predicted classes. It is especially important for multi-class classification problems where the number of possible classes is large and the model might output several plausible predictions.

- Top-1 Accuracy: Standard accuracy. True label must be the most probable prediction.
- Top-5 Accuracy: True label can be any of the top 5 predicted classes.

$$\text{Top-}k\text{ Accuracy} = \frac{1}{N}\sum_{i=1}^{N} 1[y_i \in \hat{y}_i^{(k)}]$$

Practical Example

Suppose a CNN outputs the following class probabilities for an image:

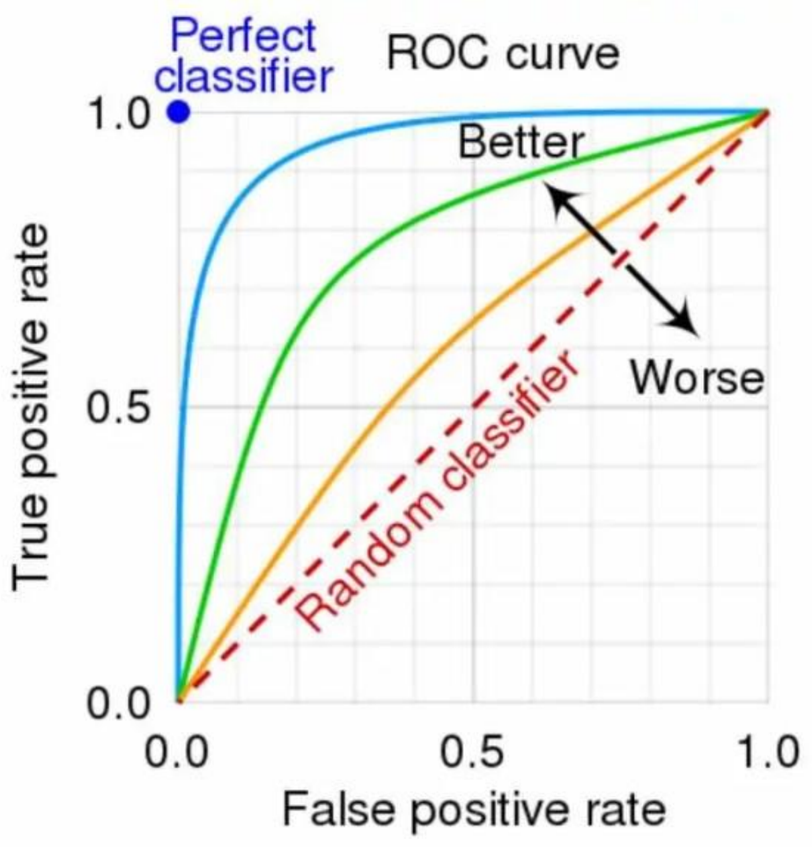
Class	Probability
Cat	0.25
Dog	0.40
Fox	0.20
Wolf	0.10
Tiger	0.05

- True label: Fox
- Top-1 prediction: Dog (incorrect)
- Top-3 predictions: Dog, Cat, Fox (correct, since Fox is in top 3)
- Top-5 predictions: All classes (correct)

ROC Curves in CNNs

ROC (Receiver Operating Characteristic) curve is a graphical plot of:

- True Positive Rate (TPR) vs False Positive Rate (FPR)
- Used to evaluate binary classification or one-vs-rest in multi-class tasks.

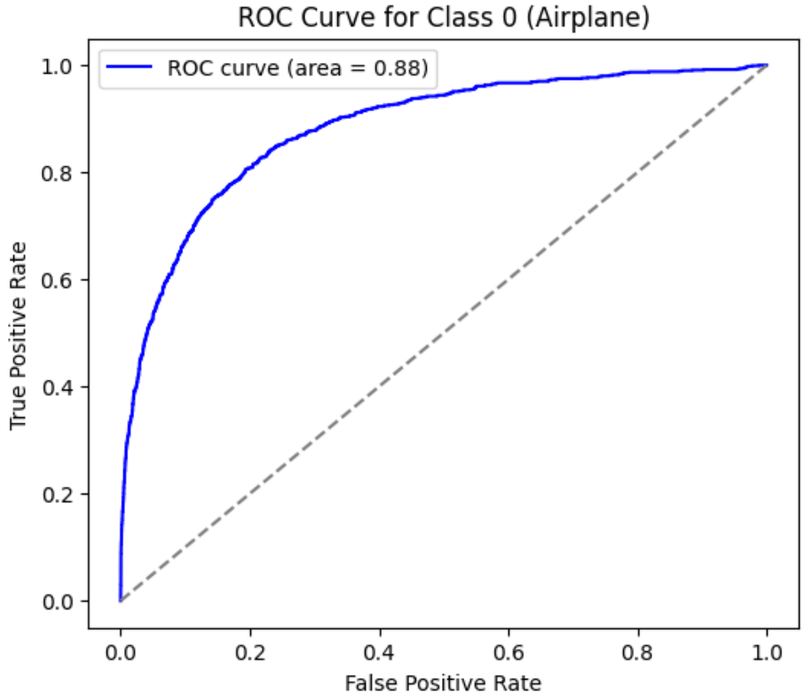


Implementing ROC

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.preprocessing import label_binarize
from sklearn.metrics import RocCurveDisplay

# Binarize labels (multi-class one-hot encoding)
y_test_bin = label_binarize(y_test, classes=range(10))

# Predict probabilities
y_score = model.predict(x_test)

# Plot ROC for class 0 (airplane) vs rest
fpr, tpr, _ = roc_curve(y_test_bin[:, 0], y_score[:, 0])
roc_auc = auc(fpr, tpr)
```

Implementing ROC (contd.)

```
plt.figure(figsize=(6,5))
plt.plot(fpr, tpr, color='blue', label=f'ROC curve
(area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Class 0 (Airplane)')
plt.legend()
plt.show()
```



Training CNNs on Small Datasets

Model 1: CNN on MNIST Dataset

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical

# Load and preprocess MNIST data
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(-1, 28, 28, 1) / 255.0
x_test = x_test.reshape(-1, 28, 28, 1) / 255.0
y_train_cat = to_categorical(y_train, 10)
y_test_cat = to_categorical(y_test, 10)

# Build CNN model
model = Sequential([ Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    MaxPooling2D(),Flatten(),Dense(64, activation='relu'),Dense(10,
    activation='softmax') ])
```

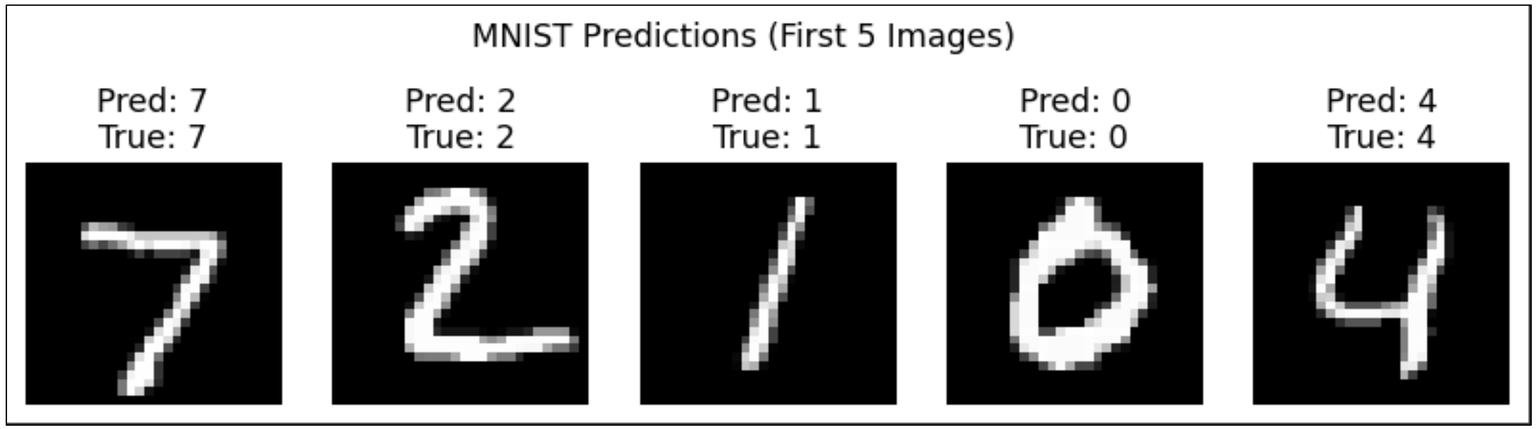
Model 1: CNN on MNIST Dataset (contd.)

```
# Compile and train
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(x_train, y_train_cat, epochs=1, batch_size=64)

# Evaluate model
loss, acc = model.evaluate(x_test, y_test_cat)
print(f"[MNIST] Test Accuracy: {acc:.4f}")

# Predict on test set
y_pred = model.predict(x_test)
y_pred_labels = np.argmax(y_pred, axis=1)
```

Model 1: CNN on MNIST Dataset (contd.)

```
plt.figure(figsize=(10,3))
for i in range(5):
    plt.subplot(1, 5, i+1)
    plt.imshow(x_test[i].reshape(28,28), cmap='gray')
    plt.title(f"Pred: {y_pred_labels[i]}\nTrue: {y_test[i][0]}")
    plt.axis('off')
plt.suptitle("MNIST Predictions (First 5 Images)")
plt.show()
```

Output:



Model 2: Code with Image Predictions

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical

# Load and preprocess CIFAR-10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train / 255.0
x_test = x_test / 255.0
y_train_cat = to_categorical(y_train, 10)
y_test_cat = to_categorical(y_test, 10)

# Build CNN model
model = Sequential([ Conv2D(32, (3,3), activation='relu', input_shape=(32,32,3)),
    MaxPooling2D(), Flatten(), Dense(64, activation='relu'),
    Dense(10, activation='softmax') ])
```

Model 2: Code with Image Predictions (contd.)

```
# Compile and train
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(x_train, y_train_cat, epochs=1, batch_size=64)

# Evaluate model
loss, acc = model.evaluate(x_test, y_test_cat)
print(f"[CIFAR-10] Test Accuracy: {acc:.4f}")

# Predict on test set
y_pred = model.predict(x_test)
y_pred_labels = np.argmax(y_pred, axis=1)

# Class names
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']
```

Model 2: Code with Image Predictions (contd.)

```
plt.figure(figsize=(12,4))
for i in range(5):
    plt.subplot(1, 5, i+1)
    plt.imshow(x_test[i])
    plt.title(f"Pred: {class_names[y_pred_labels[i]]}
      \nTrue: {class_names[y_test[i][0]]}")
    plt.axis('off')
plt.suptitle("CIFAR-10 Predictions (First 5 Images)")
plt.tight_layout()
plt.show()
```

Output:

Visualizing Class Activation Maps (CAMs)

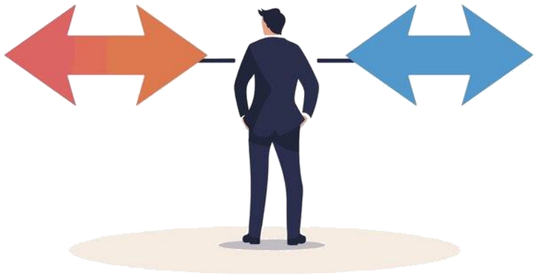
What are Class Activation Maps (CAMs)?

CAMs visualize where the model "looks" when predicting a specific class. They overlay heatmaps on the input image to highlight important regions. Especially useful in medical imaging, object detection, and deep learning explainability.

Why CAMs?

Model Interpretability

Model Comparison

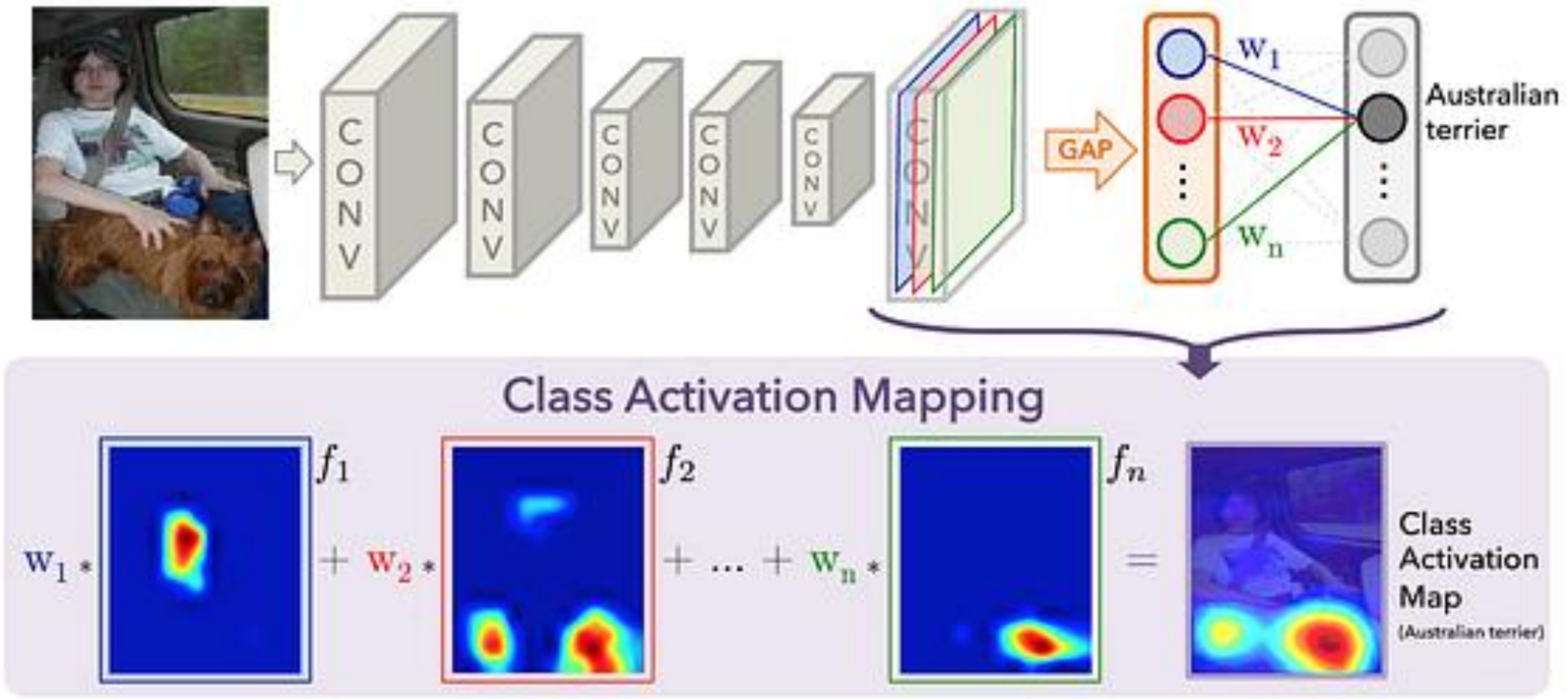


Trust & Explainability in Critical Fields

Debugging Misclassifications

BUG REPORT

Working of CAMs



Training Pre-trained CNN models on Small Dataset (Demonstration)

Note: Refer to the Module 4: Demo 1 on LMS for detailed steps.

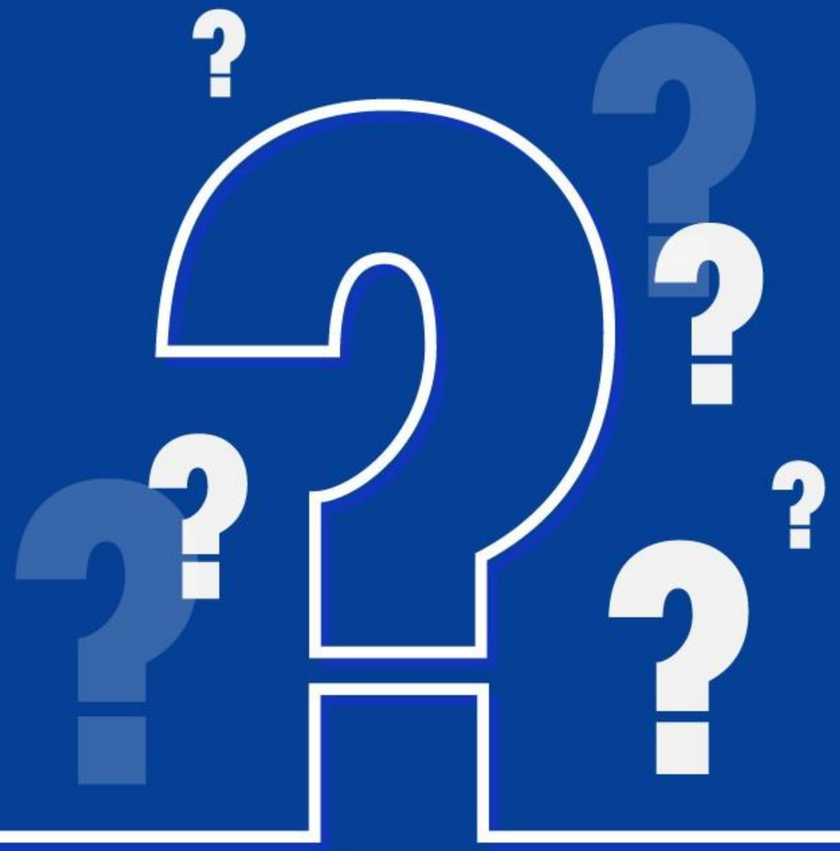
Summary

In this lesson, you have learned to:

- Illustrate AlexNet and VGGNet with their architectures.
- Apply transfer learning in CNNs.
- Use pre-trained models with Keras.
- Performance measurement with Top-k and ROC.
- Implement and train CNNs on small datasets.
- Visualizing CAMs.

edureka! a Veranda ENTERPRISE

Questions



edureka! a Veranda ENTERPRISE

Feedback

edureka! a Veranda ENTERPRISE

Thank You

For information, Please Visit our Website
www.edureka.co