



Seeing with CNNs

Deep Learning for Object Recognition

Track 2: Deep Learning (CNNs & Transfer Learning)



Introduction

1

Exploring the
CIFAR – 10
Dataset

2

Understanding
Convolutional
Neural Networks

3

Creating and
Tuning the Model

4

Exploring Transfer
Learning



CIFAR – 10 Dataset

CIFAR-10: One Sample Image per Class



- Canadian Institute for Advanced Research (CIFAR)
- 60,000 color images
- 10 different image categories
- Common Benchmark for computer vision



CIFAR – 10 Dataset



MNIST Dataset examples



n01622779 (24)



n03461385 (582)

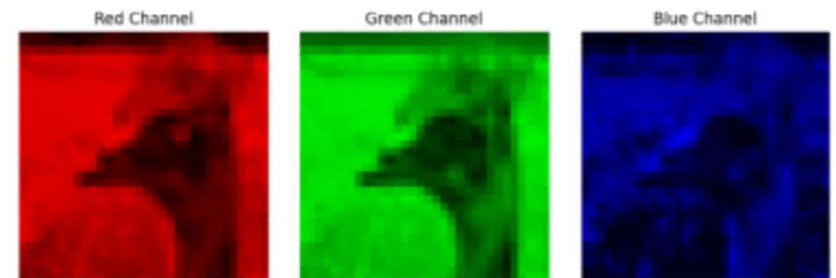
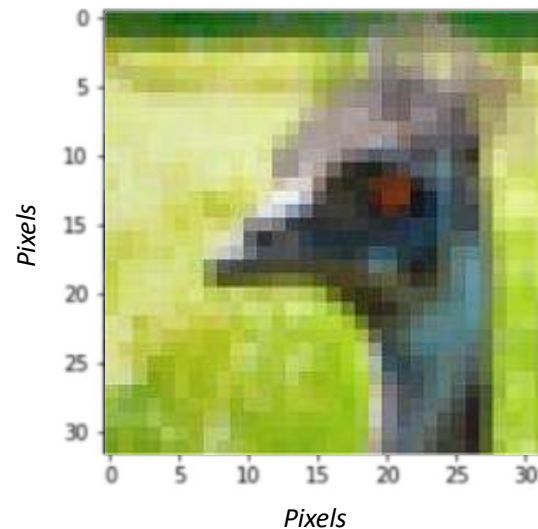


n02028035 (141)



n02086240 (155)

ImageNet Dataset examples

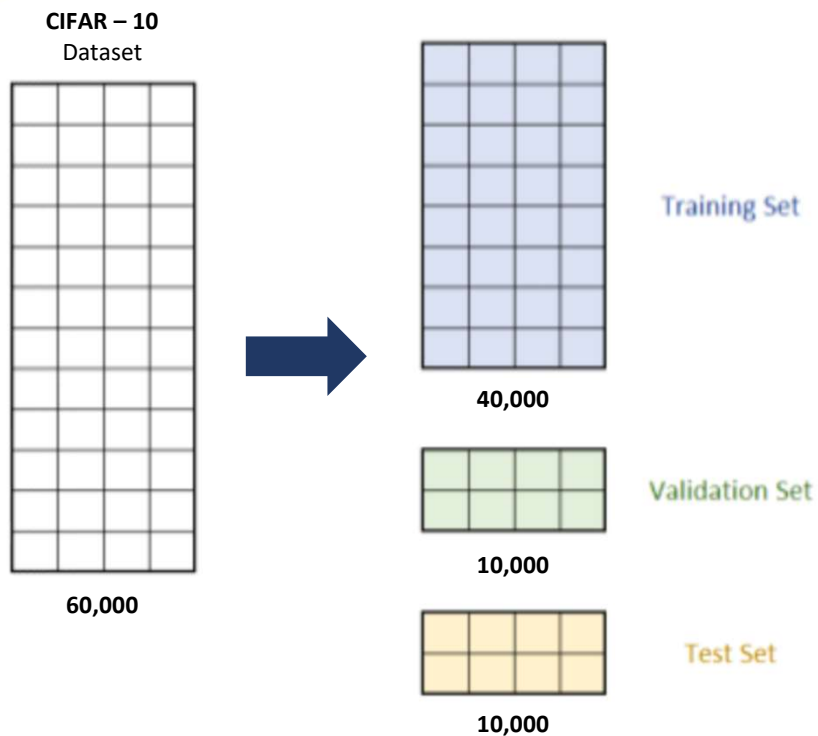


Each color image in **CIFAR-10** is 32 by 32 pixels. Each pixel color is represented by its red, green and blue intensity values.



Splitting the Dataset

How the dataset was split



Why split the dataset?

Prevents overfitting

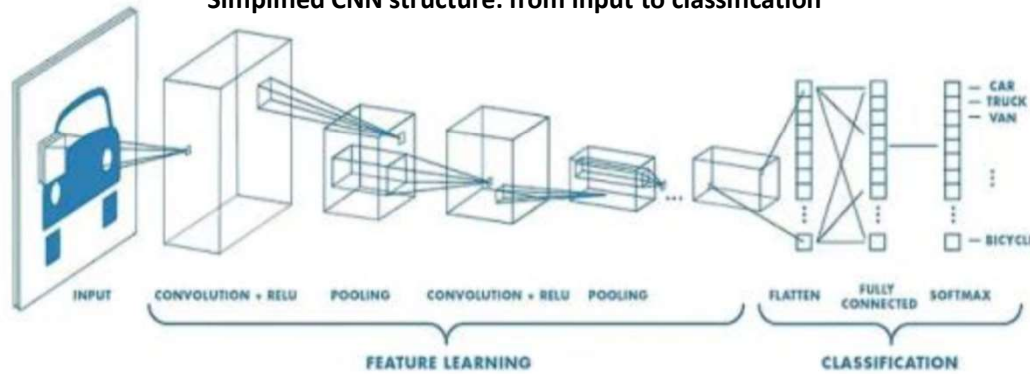
Guides parameter tuning

Ensures fair model evaluation



What is a CNN?

Simplified CNN structure: from input to classification



A **Convolutional Neural Network (CNN)** is a deep learning model that automatically learns and detects patterns within image data.

Uses of a CNN include:



Facial Recognition

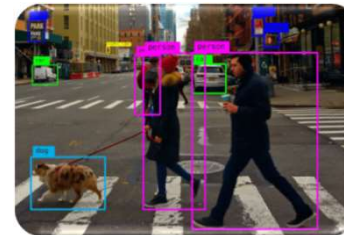
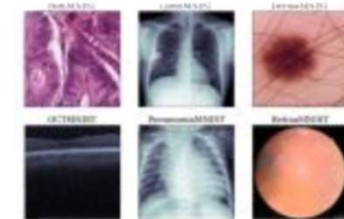


Image Detection

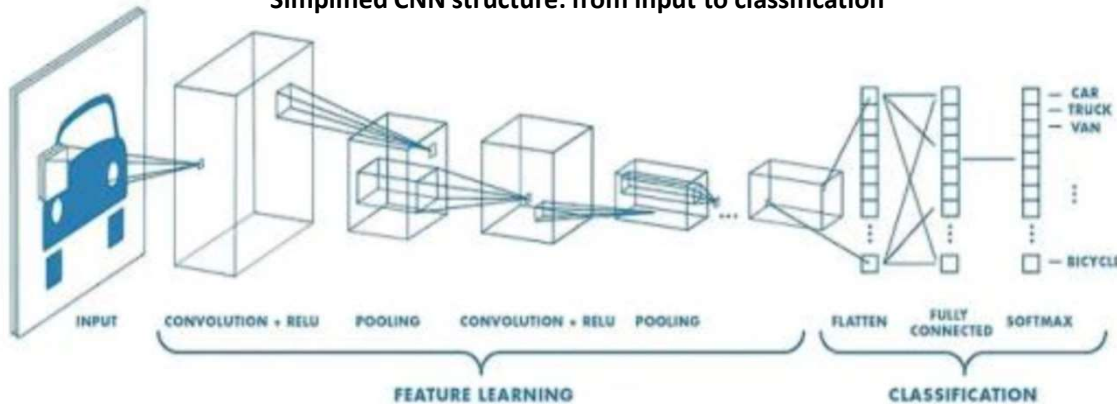


Medical Imaging

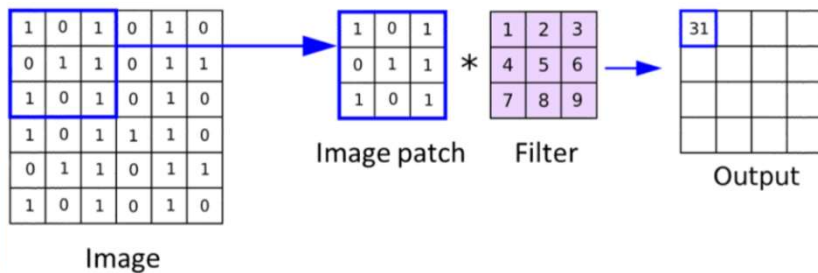


How does a CNN work?

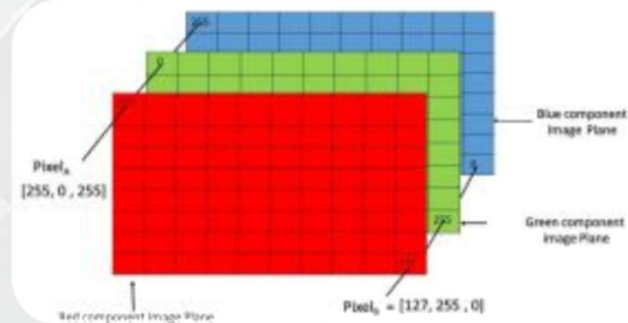
Simplified CNN structure: from input to classification



Convolutional Neural Network Process



Feature extraction using filters



Pixel color represented by 3D matrix

CNN Steps

Convolution Layers

Activation Functions

Pooling Layers

Flattening

Fully Connected Layers



Baseline CNN Model



Model was implemented in **python** using **Tensor Flow** and **Keras**

Video of the Initial training and review of the baseline model in python and Visual Studio Code

```
import numpy as np
from sklearn.metrics import classification_report
import tensorflow as tf

baseline_model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(32, 32, 3)),
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu', padding='same'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

baseline_model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.01),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

history_baseline = baseline_model.fit(
    x_train, y_train,
    validation_data=(x_val, y_val),
    epochs=5,
    batch_size=256,
    verbose=2)

test_loss, test_acc = baseline_model.evaluate(x_test, y_test, verbose=0)
print("=== Training Summary ===")
print(f"Final Training Accuracy: {history_baseline.history['accuracy'][-1]:.4f}")
print(f"Final Validation Accuracy: {history_baseline.history['val_accuracy'][-1]:.4f}")
print(f"Final Training Loss: {history_baseline.history['loss'][-1]:.4f}")
print(f"Final Validation Loss: {history_baseline.history['val_loss'][-1]:.4f}")

print("=== Test Performance ===")
print(f"Test Accuracy: {test_acc:.4f}")
print(f"Test Loss: {test_loss:.4f}")

y_pred = baseline_model.predict(x_test, verbose=0)
y_pred_classes = np.argmax(y_pred, axis=-1)
print("=== Classification Report ===")
print(classification_report(y_test, y_pred_classes, digits=2))
```

Initial Model review

```
=== Training Summary ===
Final Training Accuracy: 0.4271
Final Validation Accuracy: 0.4122
Final Training Loss: 1.5872
Final Validation Loss: 1.6402
```

```
=== Test Performance ===
Test Accuracy: 0.4125
Test Loss: 1.6294
```

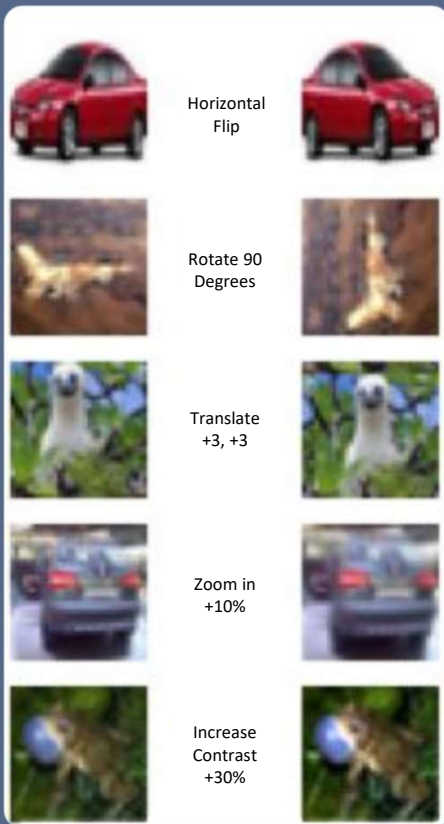
Classification Report					
	precision	recall	f1-score	support	
0	0.51	0.51	0.51	1000	
1	0.58	0.39	0.47	1000	
accuracy			0.41	10000	
macro avg	0.41	0.41	0.40	10000	
weighted avg	0.41	0.41	0.40	10000	

Baseline model's most confident misclassifications



Data Augmentation

Examples of Data Augmentation



Benefits of Data Augmentation

Increase Dataset Diversity

Improve Model Robustness

Reduces Overfitting

Simulates Real-World Variations



Tuning the Model Hyperparameters

- **Kernel Size (3×3):** Captures local image patterns effectively
- **Pooling (MaxPooling):** Reduces size, keeps key features
- **Architecture (32–64–128 + Dense 128):** More layers = richer feature extraction
- **Learning Rate (0.0005):** Controls training speed and stability
- **Epochs (25):** Number of full passes through the data
- **Batch Size (25):** Smaller batches can improve generalization
- **Activation (ReLU):** Fast and effective for deep nets

Video showing the training of the new tuned model on the augmented data

```
import math
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt

def __init__(self, data_loader):
    self.data_loader = data_loader

    # Data
    (x_train_full, y_train_full), (x_test, y_test) = self.data_loader.load_data()
    x_train_full = x_train_full.astype("float32") / 255.0
    x_test = x_test.astype("float32") / 255.0
    y_train_full = y_train_full.reshape(-1, dtype="int32")
    y_test = y_test.reshape(-1, dtype="int32")

    # Train, validation, test sets
    x_train, x_val, y_train, y_val = train_test_split(
        x_train_full, y_train_full, test_size=0.1, stratify=y_train_full, random_state=0)

    # Training
    self.model = tf.keras.Sequential([
        tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(180, 180, 3)),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(10, activation='softmax')
    ])

    self.model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

    # Training history
    history = self.model.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=25,
                             callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                                                           min_delta=0.001,
                                                                           patience=10,
                                                                           verbose=1)])

    # Evaluation
    loss, acc = self.model.evaluate(x_test, y_test, verbose=0)
    print('Test loss:', loss)
    print('Test accuracy:', acc)
```

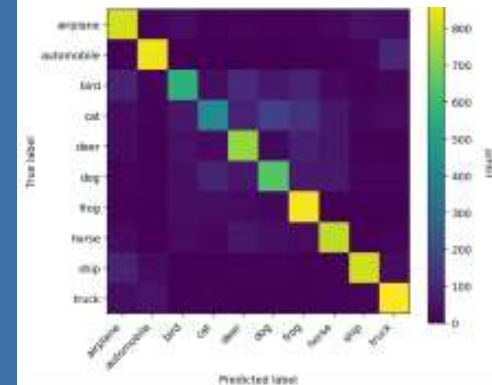
Baseline Accuracy: 41%
Tuned Accuracy: 74%

+33%

Model Test Results

accuracy			0.74	10000
macro avg	0.74	0.74	0.73	10000
weighted avg	0.74	0.74	0.73	10000

Model results Confusion Matrix



Transfer Learning

Result of using the MobileNetV2 Pretrained Model

```
=== Training Summary ===  
Final Training Accuracy: 0.9991  
Final Validation Accuracy: 0.9146  
Final Training Loss: 0.0040  
Final Validation Loss: 0.4707
```

```
=== Test Performance ===  
Test Accuracy: 0.8990  
Test Loss: 0.3296
```

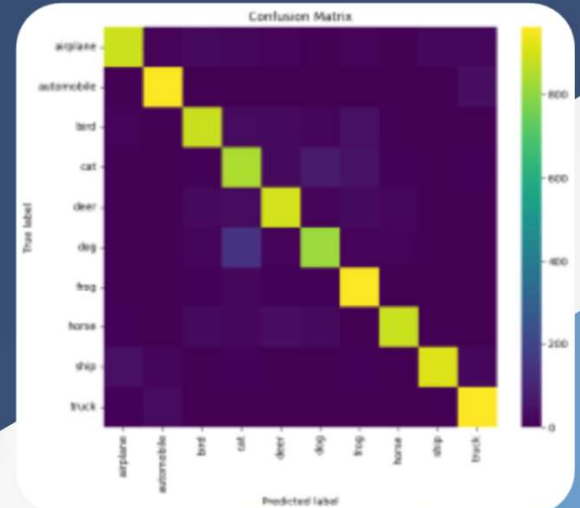
```
=== Classification Report ===  
              precision    recall  f1-score   support  
  
   airplane      0.93      0.89      0.91       1000  
  automobile      0.95      0.96      0.95       1000  
         bird      0.90      0.88      0.89       1000  
          cat      0.76      0.84      0.80       1000  
         deer      0.89      0.89      0.89       1000  
         dog      0.87      0.81      0.84       1000  
         frog      0.88      0.96      0.91       1000  
        horse      0.97      0.88      0.92       1000  
          ship      0.97      0.91      0.94       1000  
         truck      0.91      0.96      0.94       1000  
  
   accuracy                   0.90       10000  
  macro avg      0.90      0.90      0.90       10000  
weighted avg      0.90      0.90      0.90       10000
```

Most Confidant Misclassifications



Baseline Accuracy: 41%
Tuned Accuracy: 74%
Transfer Learning Accuracy: 90%

Confusion Matrix



Comparing ML Approaches

1

Traditional Machine Learning

Strengths:

- Simple and fast to implement
- Works with smaller datasets
- Low computational cost

Weaknesses:

- Requires manual feature engineering
- Struggles with complex patterns
- Lower accuracy compared to CNNs

2

Custom CNN

Strengths:

- Learns hierarchical features automatically
- High accuracy with tuning
- Effective for visual pattern recognition

Weaknesses:

- Data hungry for optimal performance
- Requires careful hyperparameter tuning
- High training time and computational cost

3

Transfer Learning

Strengths:

- High accuracy with less data and time
- Leverages powerful pretrained models
- Reduces overfitting, efficient training

Weaknesses:

- Less flexibility over learned features
- Must select a suitable pretrained model
- Slightly more complex setup than training from scratch



Conclusion

Summary of Project Scope

- 1 Understanding the CIFAR-10 dataset
- 2 Splitting the dataset
- 3 CNNs and how they work
- 4 Initializing a baseline CNN
- 5 Tuning the model's hyperparameters
- 6 Methods of augmenting the data
- 7 Transfer learning with pre-trained models
- 8 Comparing different ML approaches

Lessons Learned

1

Data preparation and splitting are essential to ensure fair and reliable evaluation.

2

CNNs automatically learn visual features, but performance depends heavily on hyperparameters and data diversity.

3

Data augmentation is crucial for improving generalisation without collecting more data.

4

Transfer learning offers a powerful shortcut, leveraging pretrained knowledge for excellent results with fewer resources



Thank you

For you time and attention!



Academic Sources

- Bergstra, J. and Bengio, Y. (2012) 'Random search for hyper-parameter optimization', *Journal of Machine Learning Research*, 13(1), pp. 281–305.
- Codebasics (2025) *Image classification using CNN (CIFAR10 dataset) | Deep Learning Tutorial 24 (TensorFlow & Python)* [YouTube video]. YouTube. Available at: <https://youtu.be/7HPwo4wnJeA>.
- Ekman, M. (2024) *Learning Deep Learning: From Perceptron to Large Language Models* [online video]. Pearson / O'Reilly. Available at: <https://learning.oreilly.com/home/>.
- Goodfellow, I., Bengio, Y. and Courville, A. (2016) *Deep Learning*. Cambridge, MA: MIT Press. (Ch. 7 – Regularization for Deep Learning).
- IBM (no date) *Convolutional Neural Networks*. Available at: <https://www.ibm.com/think/topics/convolutional-neural-networks>.
- Kohavi, R. (1995) 'A study of cross-validation and bootstrap for accuracy estimation and model selection', *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, 2(12), pp. 1137–1145.
- Kornblith, S., Shlens, J. and Le, Q.V. (2019) 'Do better ImageNet models transfer better?', *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2661–2671.
- Krizhevsky, A. (2009) *Learning multiple layers of features from tiny images*. Technical Report. Department of Computer Science, University of Toronto.
- Krizhevsky, A., Sutskever, I. and Hinton, G.E. (2012) 'ImageNet classification with deep convolutional neural networks', *Advances in Neural Information Processing Systems*, 25, pp. 1097–1105.
- LeCun, Y., Bengio, Y. and Hinton, G. (2015) 'Deep learning', *Nature*, 521(7553), pp. 436–444.
- Nair, V. and Hinton, G.E. (2010) 'Rectified linear units improve restricted Boltzmann machines', in *Proceedings of the 27th International Conference on Machine Learning (ICML)*. Omnipress.
- Pan, S.J. and Yang, Q. (2010) 'A survey on transfer learning', *IEEE Transactions on Knowledge and Data Engineering*, 22(10), pp. 1345–1359.
- Polo Club (no date) *CNN Explainer*. Available at: <https://poloclub.github.io/cnn-explainer/>.
- Prathammodi001 (no date) 'Convolutional Neural Networks for Dummies: A Step-by-Step CNN Tutorial'. *Medium*. Available at: <https://medium.com/@prathammodi001/convolutional-neural-networks-for-dummies-a-step-by-step-cnn-tutorial-e68f464d608f>.
- Shorten, C. and Khoshgoftaar, T.M. (2019) 'A survey on image data augmentation for deep learning', *Journal of Big Data*, 6(60). doi: 10.1186/s40537-019-0197-0.
- Sidana, N. (2025) 'Using Convolutional Neural Networks on the CIFAR-10 Dataset'. *Medium*, 12 February. Available at: <https://medium.com/@nsidana123/using-convolutional-neural-networks-on-the-cifar-10-dataset-1a2dc394cdd0>.
- TensorFlow (no date) *CIFAR-10 dataset*. Available at: <https://www.tensorflow.org/datasets/catalog/cifar10>.
- Towards Data Science (no date) 'Deep Learning with CIFAR-10 Image Classification'. Available at: <https://towardsdatascience.com/deep-learning-with-cifar-10-image-classification-64ab92110d79/>.
- Ultralytics (2024) *CIFAR-10 Dataset*. Available at: <https://docs.ultralytics.com/datasets/classify/cifar10/>.
- Yosinski, J., Clune, J., Bengio, Y. and Lipson, H. (2014) 'How transferable are features in deep neural networks?', *Advances in Neural Information Processing Systems*, 27, pp. 3320–3328.