December 15, 2024

# Classification of Brain-tumor images

ShivShreeram P

## Imports

The following libraries and modules are used to build and run the model:

- **NumPy**:

```
import numpy as np
```

  NumPy is a fundamental package for scientific computing in Python. It provides support for arrays, matrices, and many mathematical functions to operate on them.

- **Pandas**:

```
import pandas as pd
```

  Pandas is a powerful library for data manipulation and analysis. It provides flexible data structures like DataFrames, which are useful for handling datasets.

- **Matplotlib**:

```
import matplotlib.pyplot as plt
```

  Matplotlib is a plotting library that is used to create static, animated, and interactive visualizations in Python. It is used here for displaying images, graphs, and model performance metrics.

- **Seaborn**:

```
import seaborn as sns
```

  Seaborn is a Python data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. In this project, it's used for visualizing confusion matrices and other statistical plots.

- **OpenCV**:

```
import cv2
```

  OpenCV is a library for computer vision tasks. It provides various functions for image processing, such as loading, resizing, and transforming images.

- **glob**:

```
import glob
```

  Glob is a module used to find all the pathnames matching a specified pattern. It's particularly useful for loading image files from directories.

- **Scikit-learn (train_test_split)**:

```
from sklearn.model_selection import train_test_split
```

This function splits the dataset into training and testing sets. It is crucial for evaluating model performance.

- **TensorFlow**:

```
import tensorflow as tf
```

TensorFlow is a deep learning framework that is used for building and training machine learning models. In this project, TensorFlow is used to define and train the convolutional neural network (CNN).

- **TensorFlow Keras Layers**:

```
from tensorflow.keras import layers
```

This module provides pre-built layers for building deep neural networks. It's used to create layers like convolutional layers, dense layers, etc., for the model.

- **Scikit-learn (confusion_matrix)**:

```
from sklearn.metrics import confusion_matrix
```

The confusion matrix is a performance measurement tool for classification tasks. This helps evaluate how well the model is classifying tumor types (healthy or tumor).

- **Keras Image Preprocessing**:

```
from tensorflow.keras.preprocessing import image
```

This module provides utilities for preprocessing images, such as resizing or rescaling images to the appropriate input size for the model.

# Data Preprocessing

Data preprocessing is a crucial step in any machine learning pipeline, especially for image-based models. In this project, the preprocessing phase involves loading, resizing, normalizing images, and preparing their corresponding labels. The steps are as follows:

1. **Loading Images:**
    - Images of brain scans with and without tumors are loaded from two separate directories: `yes` (tumor) and `no` (healthy).
    - The file paths are accessed using `glob.iglob`, a method for iterating over files matching a specific pattern.

2. **Image Resizing:**

   - Each image is resized to $128 \times 128$ pixels using the `cv2.resize` function. This standardization ensures uniform input dimensions for the model.

3. **Color Conversion:**

   - The color channels of each image are reordered from BGR (default in OpenCV) to RGB using the `cv2.split` and `cv2.merge` functions. This aligns the images with the RGB format expected by TensorFlow and Matplotlib.

4. **Normalization:**

   - The pixel values of the images are scaled to the range [0, 1] by dividing by 255.0. Normalization helps accelerate the training process and reduces the likelihood of numerical instabilities.

5. **Label Assignment:**

   - Labels are assigned to the images:
     - Tumor images are labeled with 1.
     - Healthy images are labeled with 0.
   - Labels are concatenated to form a single array that corresponds to the order of the combined image data.

6. **Combining Data:**

   - All preprocessed images are combined into a single NumPy array for easy manipulation and subsequent splitting into training and testing datasets.

This preprocessing pipeline ensures the data is clean, standardized, and ready for efficient training and evaluation of the model.

## Image Set

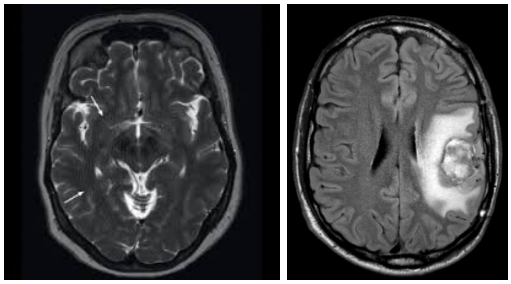

Figure 1: Sample brain scan images. Left: Healthy brain image. Right: Brain image with a tumor.

## Deep Learning Approach

To address the task of brain tumor detection, I chose to use Convolutional Neural Networks (CNNs). CNNs are particularly well-suited for image-based tasks as they excel at identifying spatial hierarchies and patterns within images, such as edges, textures, and shapes. By leveraging the feature extraction capabilities of CNNs, this approach provides a straightforward yet powerful solution to classify brain scan images into tumor or healthy categories. This method ensures efficiency and accuracy while maintaining simplicity in the implementation.

## Understanding Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a specialized class of neural networks designed specifically for processing grid-like data, such as images. CNNs have revolutionized computer vision by mimicking the way humans perceive and interpret visual information. They are built to automatically and adaptively learn spatial hierarchies of features from input images.

### How CNNs Work

A CNN processes images through a series of layers, each designed to extract and learn features of increasing complexity. The key components of a CNN are as follows:

- **Convolutional Layers:** These layers apply filters (kernels) to the input image, producing feature maps that highlight specific patterns such as edges, textures, or objects.

- **Pooling Layers:** These layers reduce the dimensionality of feature maps by summarizing information, making the model more computationally efficient and robust to variations in the image.
- **Fully Connected Layers:** In the final stages, the extracted features are flattened and passed through fully connected layers for classification.
- **Activation Functions:** Non-linear functions such as ReLU (Rectified Linear Unit) introduce non-linearity, enabling the network to learn complex mappings.

## Example: Feature Extraction with Convolutional Layers

Convolutional layers apply small filters to the image, detecting patterns such as edges or corners. This process is illustrated below:

## Pooling: Reducing Dimensionality

Pooling layers condense information in feature maps. Max pooling, for example, selects the highest value from a feature map region, as shown below:

## Why CNNs Are Ideal for Image Classification

CNNs are designed to handle the challenges of image data:

- **Locality and Translation Invariance:** CNNs focus on local patterns, which are combined at higher layers to form more complex features. This makes them invariant to minor translations or rotations.
- **Parameter Efficiency:** By sharing weights across the spatial dimensions, CNNs significantly reduce the number of parameters compared to fully connected networks.
- **Hierarchical Learning:** The layered structure allows CNNs to learn low-level features (e.g., edges) in early layers and high-level features (e.g., shapes or objects) in later layers.

This capability makes CNNs an excellent choice for solving the problem of brain tumor classification.

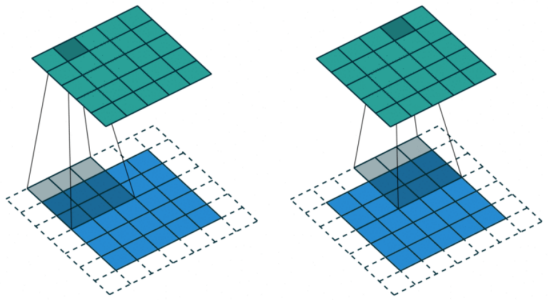

Figure 2: Illustration of a convolution operation. A filter slides over the input image to produce a feature map highlighting specific patterns.

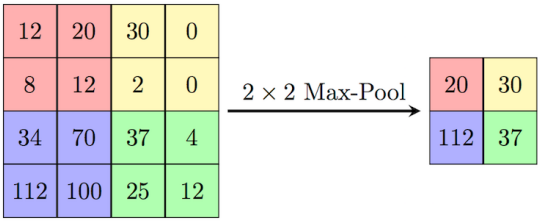


Figure 3: Example of max pooling. The maximum value from each region is retained, reducing the feature map size while preserving important information.

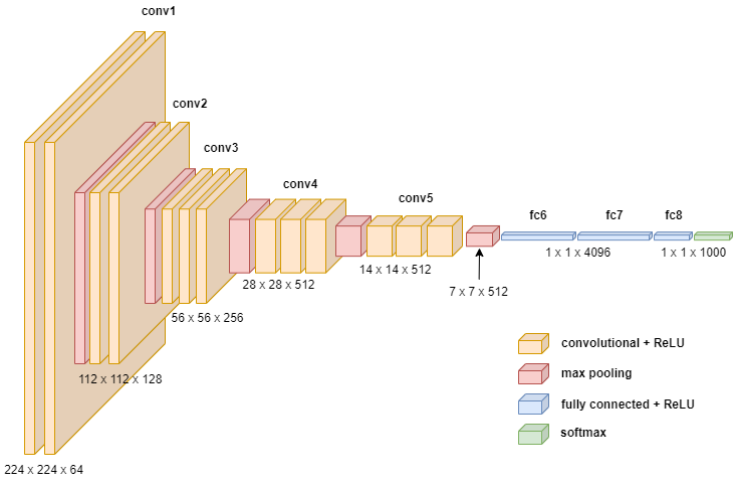


Figure 4: Typical CNN architecture for image classification tasks. Input images pass through convolutional, pooling, and fully connected layers for classification.

## Brain Tumor Classification Model

The dataset used for the brain tumor classification model was sourced from Kaggle. The dataset contains brain scan images of two classes: tumors and non-tumors. Specifically, the dataset consists of:

- 98 images with tumors.
- 154 images without tumors (healthy).

To train and evaluate the model, the dataset was divided into three sets:

- **Training Set (70%):** This set is used for training the model.
- **Validation Set (20%):** This set is used to tune the model and check its performance during training.
- **Test Set (10%):** This set is used to evaluate the model's final performance.

The split resulted in the following number of images for each class:

- **Tumor images:**
  - 68 images for training.
  - 20 images for validation.
  - 10 images for testing.
- **Non-tumor images:**
  - 108 images for training.
  - 31 images for validation.
  - 15 images for testing.

The dataset is unbalanced, with more non-tumor images than tumor images. During training, techniques like data augmentation or class balancing may be applied to handle this imbalance effectively.

## CNN Model Overview

The following describes the architecture of the SimpleCNN model used for brain tumor classification, including the layer types, activation functions, and the formulas for calculating the output size at each step.

## Layer-wise Details

**Input Layer:** The input to the model is an image of size $128 \times 128$ with 3 color channels (RGB), resulting in a shape of $(128, 128, 3)$.

**Convolutional Layer 1 (conv1):** - Filter Size: $3 \times 3$, Number of Filters: 32 - Activation Function: ReLU - Padding: Same (no padding, implicitly) - Stride: 1 - Output Size: The formula for the output size of a convolutional layer is:

$$\text{Output size} = \frac{\text{Input size} - \text{Kernel size} + 2 \times \text{Padding}}{\text{Stride}} + 1$$

Substituting the values for this layer:

$$\text{Output size} = \frac{128 - 3 + 2 \times 0}{1} + 1 = 126$$

Thus, the output shape of this layer is $(126, 126, 32)$.

**Max Pooling Layer 1 (pool1):** - Pool Size: $2 \times 2$ - Stride: 2 - Output Size: The formula for max pooling is:

$$\text{Output size} = \frac{\text{Input size} - \text{Pool size}}{\text{Stride}} + 1$$

For this layer:

$$\text{Output size} = \frac{126 - 2}{2} + 1 = 63$$

Thus, the output shape of this layer is $(63, 63, 32)$.

**Convolutional Layer 2 (conv2):** - Filter Size: $3 \times 3$, Number of Filters: 64 - Activation Function: ReLU - Padding: Same - Stride: 1 - Output Size: Using the same formula as for conv1, we get:

$$\text{Output size} = \frac{63 - 3 + 2 \times 0}{1} + 1 = 61$$

Thus, the output shape of this layer is $(61, 61, 64)$.

**Max Pooling Layer 2 (pool2):** - Pool Size: $2 \times 2$ - Stride: 2 - Output Size: Using the pooling formula:

$$\text{Output size} = \frac{61 - 2}{2} + 1 = 30$$

Thus, the output shape of this layer is $(30, 30, 64)$.

**Convolutional Layer 3 (conv3):** - Filter Size: $3 \times 3$, Number of Filters: 128 - Activation Function: ReLU - Padding: Same - Stride: 1 - Output Size: Using the convolution formula:

$$\text{Output size} = \frac{30 - 3 + 2 \times 0}{1} + 1 = 28$$

Thus, the output shape of this layer is $(28, 28, 128)$.

**Max Pooling Layer 3 (pool3):** - Pool Size: $2 \times 2$ - Stride: 2 - Output Size: Using the pooling formula:

$$\text{Output size} = \frac{28-2}{2} + 1 = 14$$

Thus, the output shape of this layer is $(14, 14, 128)$.

**Flatten Layer:** This layer flattens the 3D output from the previous pooling layer into a 1D vector. The output shape becomes $14 \times 14 \times 128 = 25088$.

**Fully Connected Layer 1 (fc1):** - Number of Neurons: 128 - Activation Function: ReLU - Output Shape: $(128, )$ - Number of Parameters: $25088 \times 128 + 128 = 3,212,992$.

**Dropout Layer 1:** - Dropout Rate: 0.5 - Output Shape: $(128, )$

**Fully Connected Layer 2 (fc2):** - Number of Neurons: 64 - Activation Function: ReLU - Output Shape: $(64, )$ - Number of Parameters: $128 \times 64 + 64 = 8,256$.

**Dropout Layer 2:** - Dropout Rate: 0.5 - Output Shape: $(64, )$

**Fully Connected Layer 3 (fc3):** - Number of Neurons: 1 - Activation Function: Sigmoid - Output Shape: $(1, )$ - Number of Parameters: $64 \times 1 + 1 = 65$.

**Total Parameters:** The total number of parameters in the model is the sum of all the parameters from each layer, which is $3,314,545$.

# Model Compilation and Training Process

After defining the CNN model, we compile it and set up the training process with the following components:

## Model Compilation

**Optimizer:** We use `Adam` with a learning rate of 0.0001. Adam adapts the learning rate for each parameter, improving convergence:

$$\theta_t = \theta_{t-1} - \eta \frac{m_t}{\sqrt{v_t} + \epsilon}$$

where $m_t$ and $v_t$ are moving averages of gradients.

**Loss Function:** The model uses `BinaryCrossentropy` for binary classification (tumor vs non-tumor). It is defined as:

$$\text{Binary Cross-Entropy} = -\left(y \log(p) + (1-y) \log(1-p)\right)$$

**Metrics:** `Accuracy` is used to measure the model's performance during training:

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}}$$

## Callbacks

**Learning Rate Scheduler:** `ReduceLROnPlateau` reduces the learning rate by a factor of 0.5 if the validation loss doesn't improve for 5 epochs.
**Early Stopping:** `EarlyStopping` halts training if the validation loss does not improve for 15 epochs, preventing overfitting.

### Training the Model

We use the `fit` method with the following settings:

- Training data: `X_train` and `y_train`
- Batch size: 32
- Up to 100 epochs, with early stopping
- Validation data: `X_val` and `y_val`
- Callbacks: `lr_scheduler` and `early_stopping`

The training returns a `history` object to track performance.

## Model Outputs

### Insights on Model Training and Validation Performance

- **Training Accuracy and Loss:** The model shows consistent improvement in training accuracy, starting with 93.16% in the first epoch and reaching nearly 99.42% by the 16th epoch. Similarly, training loss decreases steadily, indicating that the model is learning effectively from the data.
- **Validation Accuracy and Loss:** There are some fluctuations in validation accuracy and loss throughout the training process. For example, in epochs 2, 7, and 12, the validation accuracy drops, even as the training accuracy improves. This indicates that the model initially struggles to generalize well on the validation data.
- **Learning Rate Adjustment:** The use of a learning rate scheduler (`ReduceLROnPlateau`) helped stabilize the training process. As the validation loss plateaus, the learning rate is reduced, allowing the model to fine-tune its weights and improve generalization. This adjustment is visible in the later epochs, where the model's performance stabilizes and continues to improve.
- **Validation Fluctuations:** The validation performance fluctuates, particularly in the early stages (e.g., epoch 2, epoch 7). These fluctuations can be attributed to factors such as:
  - **Learning Rate:** The initial learning rate might have been too high, causing overshooting. The learning rate reduction helped mitigate this.
  - **Model Overfitting/Underfitting:** Early fluctuations can also be a result of the model overfitting to the training data, but callbacks like early stopping and learning rate reduction helped control this issue.

- **Data Variability:** The validation data might have some inherent noise, causing occasional performance dips.

- **Final Outcome:** Despite the early fluctuations, the model stabilizes towards the end of training, with validation accuracy improving steadily. The use of early stopping helped prevent overfitting, ensuring that the model generalized well to unseen data.

**In summary,** the model shows strong performance in both training and validation, with some initial validation fluctuations being addressed by the learning rate scheduler and early stopping mechanisms. The overall trend is positive, and the model demonstrates excellent learning and generalization capabilities.

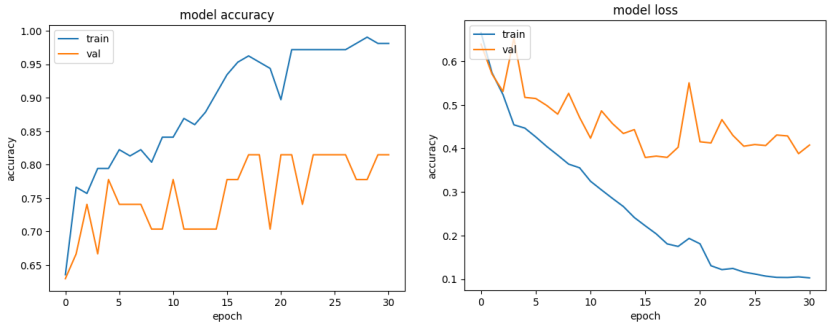


Figure 5: Training vs Validation Accuracy (Left) and Training vs Validation Loss (Right)

## Visual Analysis

**Training Data Analysis:** The confusion matrix for the training data is as follows:

| Predicted | Predicted Healthy | Predicted Tumor |
|---|---|---|
| **Actual Healthy** | 52 | 2 |
| **Actual Tumor** | 2 | 51 |

From the confusion matrix, we can calculate the following metrics:

- **True Positives (TP):** 51, representing the correctly predicted tumor cases.

- **False Positives (FP):** 2, representing the incorrectly predicted tumor cases.

- **False Negatives (FN):** 2, representing the incorrectly predicted healthy cases.

- **True Negatives (TN):** 52, representing the correctly predicted healthy cases.
- **Accuracy:** $\frac{TP+TN}{TP+FP+FN+TN} = \frac{51+52}{51+2+2+52} = 0.976$ or 97.6%.
- **Precision for Tumor:** $\frac{TP}{TP+FP} = \frac{51}{51+2} = 0.962$ or 96.2%.
- **Recall for Tumor:** $\frac{TP}{TP+FN} = \frac{51}{51+2} = 0.962$ or 96.2%.
- **Specificity for Healthy:** $\frac{TN}{TN+FP} = \frac{52}{52+2} = 0.963$ or 96.3%.

**Validation Data Analysis:** The confusion matrix for the validation data is as follows:

| **Predicted** | **Predicted Healthy** | **Predicted Tumor** |
|---|---|---|
| **Actual Healthy** | 13 | 4 |
| **Actual Tumor** | 2 | 8 |

From the confusion matrix, we calculate the following metrics:

- **True Positives (TP):** 8, representing the correctly predicted tumor cases.
- **False Positives (FP):** 4, representing the incorrectly predicted healthy cases.
- **False Negatives (FN):** 2, representing the incorrectly predicted tumor cases.
- **True Negatives (TN):** 13, representing the correctly predicted healthy cases.
- **Accuracy:** $\frac{TP+TN}{TP+FP+FN+TN} = \frac{8+13}{8+4+2+13} = 0.80$ or 80%.
- **Precision for Tumor:** $\frac{TP}{TP+FP} = \frac{8}{8+4} = 0.667$ or 66.7%.
- **Recall for Tumor:** $\frac{TP}{TP+FN} = \frac{8}{8+2} = 0.80$ or 80%.
- **Specificity for Healthy:** $\frac{TN}{TN+FP} = \frac{13}{13+4} = 0.765$ or 76.5%.

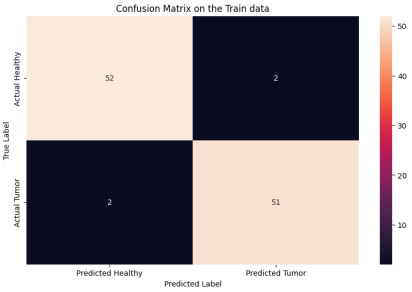


Figure 6: Confusion Matrix: Training data

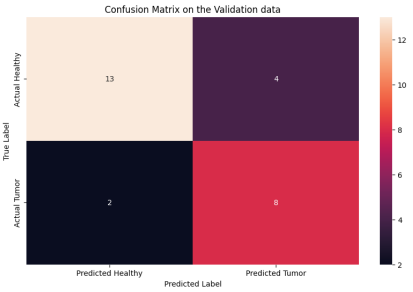


Figure 7: Confusion Matrix: Validation data

# Model Testing

**Conclusion:** The model shows strong performance on the training data, accurately classifying most instances. Although the validation data shows some misclassification, the model still demonstrates reliable generalization.

## Theory Behind Model Testing

Model testing on unseen data is a critical step in evaluating the real-world performance of a machine learning model. The purpose of testing is to assess the model's ability to generalize to data that it has not seen before, which helps determine if the model can make accurate predictions on new, previously unseen data. This step is essential to ensure that the model does not just memorize the training data (overfitting), but instead learns meaningful patterns that can be applied to new inputs.

In this case, after training the model on the available data and validating its performance, we evaluate it on a separate test set. This allows us to estimate how well the model will perform when deployed in real-world scenarios where the input data will not always match the training set.

Additionally, a classification threshold of 0.5 is used to decide the prediction outcome: if the model's output is less than 0.5, it is classified as 'Healthy,' and if it is greater than or equal to 0.5, it is classified as 'Tumor.' This threshold is commonly used in binary classification tasks where outputs are probabilities between 0 and 1.

## Testing on Unseen Data

The model was tested on unseen data, and below are the images showing the correct predictions made by the model on this test set. The classification threshold of 0.5 was applied to decide the label of the predictions.

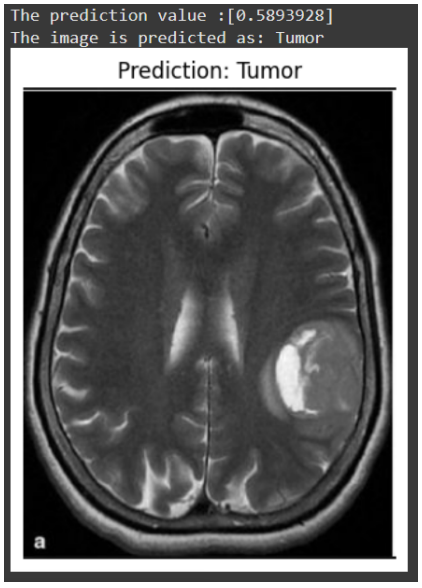


Figure 8: Correct Prediction 1 - The model correctly identified the tumor as positive (Prediction $\geq 0.5$).


Figure 9: Correct Prediction 2 - The model correctly identified healthy tissue as negative (Prediction $<$ 0.5).

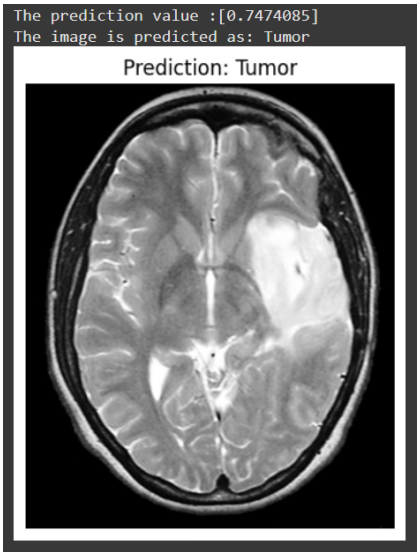


Figure 10: Correct Prediction 3 - Another example of correct tumor classification.

The images above show the correct predictions made by the model on the test data, where the threshold of 0.5 was applied for classification. If the model's predicted probability for a tumor was greater than or equal to 0.5, it was classified as 'Tumor,' otherwise it was classified as 'Healthy.'

### Significance of Model Testing

Testing on unseen data is a crucial part of the model evaluation process as it provides an insight into how the model might perform on real-world, unseen examples. By validating with unseen data, we ensure that the model can make accurate predictions beyond the training and validation datasets. This is key to determining the model's practical usability and robustness.

The correct predictions on the unseen data, as shown in the figures above, indicate that the model has successfully generalized from the training data to new, unknown cases, showcasing its reliability and effectiveness.

## References


[1] Fig. 2 - *Intelligent Systems course, FutureLearn.* Available: `https://www.futurelearn.com/info/courses/intelligent-systems/0/steps/245920`.

[2] Fig. 3 - *Max Pooling, Papers with Code.* Available: `https://paperswithcode.com/method/max-pooling`.

[3] Fig. 4 - *Deep Dive into Convolutional Neural Networks, LinkedIn.* Available: `https://www.linkedin.com/pulse/deep-dive-convolutional-neural-networks-cnns-linkedin-subramanian`.

[4] Credits to *ML Dawn (YouTube Channel).*