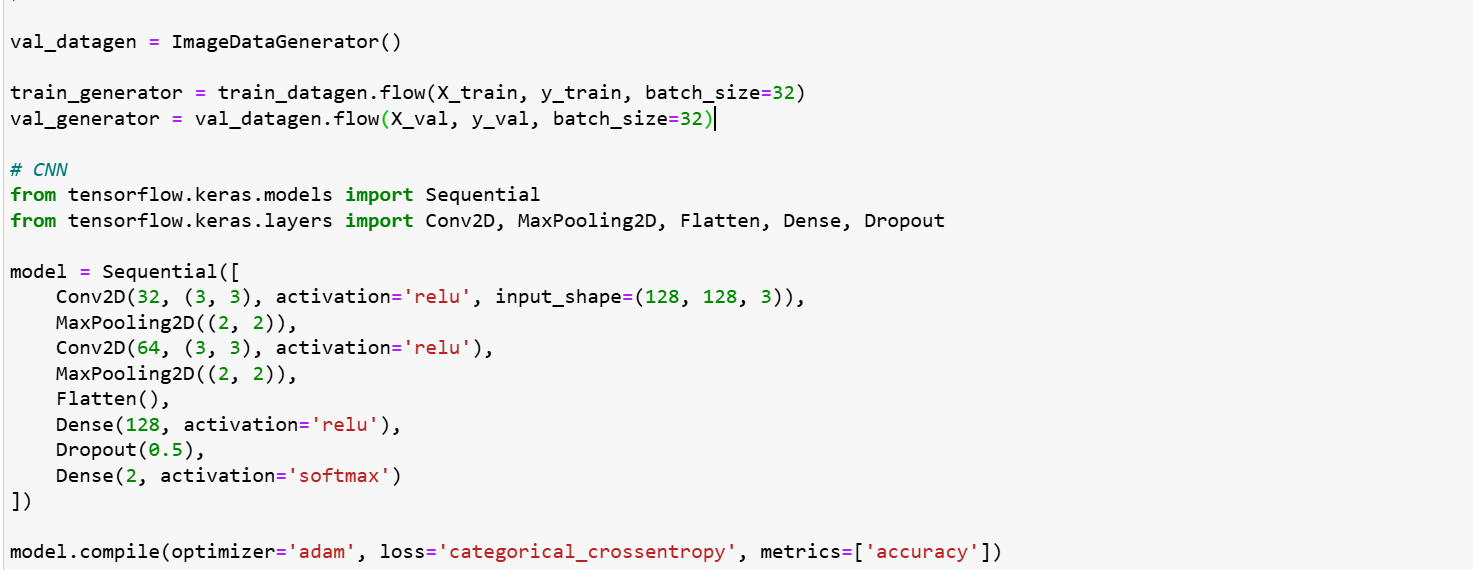
**Writeup of the Convolutional Neural Network (CNN) Model for Brain Tumor Detection**

**Overview**

The model is designed to classify images into two categories: 'no' tumor and 'yes' tumor. The process includes data loading, preprocessing, model building, training, and evaluation.

1. **Data Loading and Preprocessing**
   * **Data Directory Structure**:
     + The data is organized in a directory with two subdirectories: 'no' and 'yes', each containing images corresponding to the absence or presence of brain tumors.
   * **Image Loading**:
     + Images are loaded from these directories using the PIL library. Each image is resized to a consistent size of 128x128 pixels and converted to RGB format.
     + The images are resized because larger images increases the computational efficiency requiring more memory. Resizing images to a smaller size like 128x128 pixels allows the model to process more images per unit time during training, speeding up the training process.
     + We have the MRI grayscale images where each pixel intensity represents the level of gray in that part of the image.
     + Convolutional Neural Networks (CNNs) commonly expect input images to have three color channels (Red, Green, Blue) due to their design for processing color images.
     + The conversion process involves transforming each grayscale MRI image into a three-channel RGB format. This is done by replicating the grayscale intensity values across all three color channels (R, G, B), resulting in a pseudo-color image where R = G = B = grayscale intensity.
     + Here, in the code it is done by using img = img.convert('RGB').
   * **Data and Labels**:
     + Loaded images are converted to numpy arrays and stored in the data list, while corresponding labels (0 for 'no' and 1 for 'yes') are stored in the labels list.
2. **Data Normalization and Conversion**
   * **Normalization**:
     + Pixel values of images are scaled from the range [0, 255] to [0, 1] by dividing by 255.0. This helps in faster and more stable training of the neural network.
     + Helps in faster convergence during training as the gradients are more stable and the weights are updated more effectively.
     + By normalizing the data to [0, 1], we ensure that the input values are compatible with the activation functions used (e.g., ReLU, Softmax), which typically perform better with inputs in this range.
   * **Conversion to Arrays**:
     + The lists data and labels are converted to numpy arrays for compatibility with TensorFlow operations.
3. **One-Hot Encoding of Labels**
   * Labels are converted to one-hot encoded vectors using the to\_categorical function. This is necessary for the categorical cross-entropy loss function used during training.
4. **Train-Validation Split**
   * The dataset is split into training and validation sets using an 80-20 split. This allows the model to be trained on one portion of the data while its performance is validated on another portion.
5. **Data Augmentation**
   * **Training Data Augmentation**:
     + The ImageDataGenerator class is used to apply random transformations to the training images (rotation, width/height shift, horizontal flip, zoom) to artificially increase the diversity of the training data.
     + **rotation\_range=20**: This parameter randomly rotates images by a degree within the specified range (0-20 degrees). This helps in making the model robust to slight rotations, simulating real-world conditions where an image might not always be perfectly aligned.
     + **width\_shift\_range=0.2** and **height\_shift\_range=0.2**: These parameters randomly shift the image horizontally and vertically by up to 20% of the total width and height. This helps the model generalize better by making it less sensitive to the position of the features within the image.
     + **horizontal\_flip=True**: Randomly flips images horizontally. This helps the model learn that the presence of a tumor is not dependent on the left-right orientation.
     + **zoom\_range=0.2**: Randomly zooms into images by up to 20%. This makes the model more robust to variations in scale.



* + **Validation Data Augmentation**:
    - Only basic rescaling is applied to validation data without any additional augmentations.
    - Batch Size signifies the number of training samples processed before the model's weights are updated.

1. **Model Architecture**
   * **Sequential Model**
     + The Sequential model allows to create models layer-by-layer in a linear fashion. Layers are added sequentially, and the data flows from one layer to the next without branching or skipping layers.
   * **Convolutional Layers**: Conv2D
     + Convolution involves sliding a small matrix (called a filter or kernel) over the input data. At each position, the filter computes a dot product between its weights and the corresponding input values.
     + Two convolutional layers are added, each followed by ReLU activation and max pooling. The first layer has 32 filters and the second has 64 filters, both using 3x3 kernels.
     + Detects features such as edges, textures, and shapes.

* Max Pooling Layer (MaxPooling2D):
* **MaxPooling2D**: This layer performs max pooling, which reduces the spatial dimensions of the input.
* **(2, 2)**: The size of the pooling window. This means a 2x2 window slides over the input, and the maximum value within each window is taken.
  + **Flatten Layer**:
    - This layer converts the 2D matrix of features into a 1D vector. This is necessary to transition from convolutional layers to fully connected layers.
  + **Dense Layers**:
    - A dense layer with 128 neurons and ReLU activation is added, followed by a dropout layer with a 50% dropout rate to prevent overfitting.
  + **Output Layer**:
    - The output layer has 2 neurons (one for each class) with Softmax activation to convert the logits to class probabilities.

**The Activation Functions**

ReLU (Rectified Linear Unit)

* ReLU: Defined as 
* Purpose: Introduces non-linearity into the model. It helps the network learn more complex patterns by enabling the learning of non-linear decision boundaries.
* Advantages: Computationally efficient, helps mitigate the vanishing gradient problem, and speeds up training.

Softmax

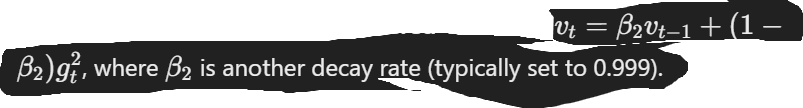
* Softmax: Converts raw logits (unscaled values) into probabilities.
* Formula: 
* Purpose: Used in the output layer for multi-class classification problems. It ensures that the output probabilities sum to 1, making it easier to interpret the model's predictions as probabilities.

1. **Model Compilation**
   * An optimizer in the context of neural networks and machine learning is an algorithm or method used to adjust the parameters of the model (e.g., weights and biases) in order to minimize the loss function.
   * The model is compiled using the Adam optimizer, categorical cross-entropy loss, and accuracy as the evaluation metric.

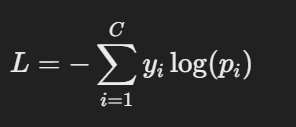
**The Adam optimizer**

The Adam optimizer is an advanced optimization algorithm that is widely used in training neural netwoks. It combines the benefits of two other popular optimization techniques: **AdaGrad and RMSProp.**

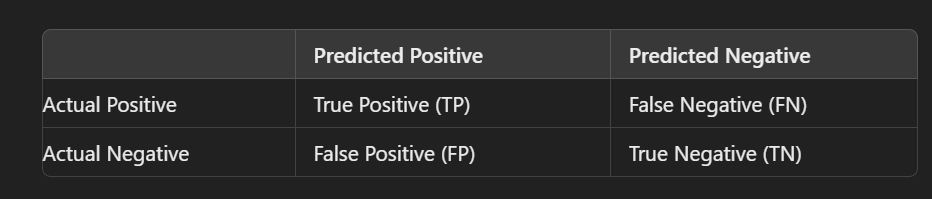
**How Adam Works ?**

* **Gradient Calculation**: For each parameter​, Adam computes the gradient of the loss function with respect to the parameter.
* **First Moment Estimate (Mean)**: , where beta1 is the decay rate(typically set to 0.9).
* **Second Moment Estimate (Uncentered Variance)**: 
* **Bias Correction**: To counteract the biases introduced by initializing the first and second moment estimates to zero, Adam includes bias correction terms: 
* **Parameter Update**: Finally, the parameters are updated using the corrected estimates:  where α is the learning rate and ϵ is a small constant to prevent division by zero.

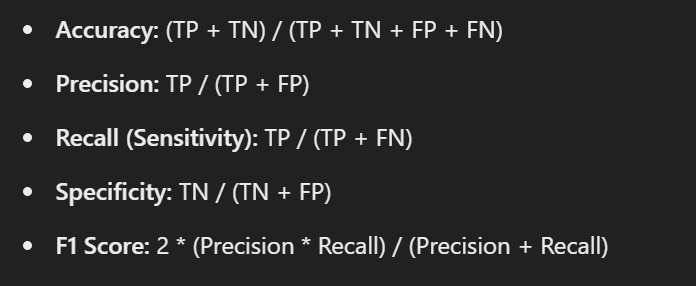
**Categorical Cross-Entropy:**

* Measures the difference between two probability distributions: the true distribution (actual labels) and the predicted distribution (output of the model).
* It quantifies the loss by comparing the predicted class probabilities with the actual class labels.
* For a single sample, the categorical cross entropy is given by, where C is the number of classes, yi​ is a binary indicator (0 or 1) if class iii is the correct classification, and pi is the predicted probability for class i.

1. **Model Training**
   * The model is trained using the training data generator for 20 epochs, with validation performed on the validation data generator. The training process involves iterative updates of the model weights based on the training data and evaluation on the validation data.
   * **epochs** refer to the number of times the entire dataset is passed forward and backward through the neural network during training. Each pass of the entire dataset through the network represents one epoch.
   * **Batch Size vs. Epochs:**
     + **Batch Size**: Refers to the number of samples processed before the model is updated. The number of iterations per epoch depends on the batch size (e.g., with a batch size of 32, each epoch includes number of samples / batch size iterations).
2. **Model Evaluation and Metrics**
   * **Validation Accuracy**:
     + The model's accuracy on the validation set is evaluated.
     + Accuracy = Total Number of Predictions/Number of Correct Predictions​
   * **Predictions**:
     + Predictions are made on the validation set to assess the model's performance.
   * **Classification Report**:
     + A classification report is generated to show precision, recall, F1-score, and support for each class.
   * **Confusion Matrix**:
     + A confusion matrix is plotted to visualize the performance of the model in distinguishing between the two classes.



* **True Positive (TP):** The number of instances where the model correctly predicted the positive class.
* **False Negative (FN):** The number of instances where the model incorrectly predicted the negative class when it was actually positive.
* **False Positive (FP):** The number of instances where the model incorrectly predicted the positive class when it was actually negative.
* **True Negative (TN):** The number of instances where the model correctly predicted the negative class.



* + **ROC Curve and AUC**:
    - The ROC curve is plotted, and the AUC score is calculated to evaluate the model's ability to distinguish between the two classes across different threshold values.
    - **ROC Curve:**
    - The Receiver Operating Characteristic (ROC) curve is a graphical representation of a classifier's performance across all classification thresholds. It plots the True Positive Rate (TPR) against the False Positive Rate (FPR).
    - **True Positive Rate (TPR):** Also known as recall or sensitivity, it is calculated as TP / (TP + FN).
    - **False Positive Rate (FPR):** It is calculated as FP / (FP + TN).
    - An ROC curve shows the trade-off between TPR and FPR across different threshold settings. A model with good predictive power will have a curve that hugs the top-left corner of the plot.
    - **AUC:**
    - The Area Under the ROC Curve (AUC) quantifies the overall ability of the model to discriminate between positive and negative classes. It ranges from 0 to 1:
    - **AUC = 1:** Perfect model.
    - **AUC = 0.5:** Model has no discriminative power (equivalent to random guessing).
    - **AUC < 0.5:** Model performs worse than random guessing.

**Interpretation:**

* A high AUC indicates a model that can effectively distinguish between the positive and negative classes.
* The ROC curve helps in choosing the optimal threshold that balances sensitivity (recall) and specificity.