ALGORITHMS AND WORKFLOW

Setup:
Import tensorflow and other necessary libraries. Download the directory containing multiple subdirectories for each of the 6 classes. These directories contain all the images needed for training and validation.Can be done by mounting drive in Google Coab.
Data is successfully downloaded. Now we have to create datasets from these images and classes.

DataSplitting:
For custom datasets, split into training, validation, and test sets.
Data preprocessing is done to all these datasets.The validation split refers to the portion of your training data that is set aside during the training process to evaluate the model's performance on data it hasn't seen during training. This helps in assessing how well model generalizes to new, unseen data.We may use validation split of 0.2(20% of the images will be used for validation, and 80% will be used for training)or k fold cross validation where dataset is divided into k subsets ,model is trained on k-1 subsets and validated on the remaining subset(this is repeated k times taking each subset as the test set with respect to other k-1 sets).
Testing is done using an image not present in training and validation sets.

Data Preprocessing:
(Either apply preprocessing layer to dataset or add it as part of the model)
Images are in different shapes and sizes and come from different sources.
We fix the batch size. Batch size determines how many samples are fed into the model at once. Choosing the right batch size can affect training speed and performance. Smaller batch sizes may lead to more noisy gradient estimates while larger batch sizes can be more stable but require more memory.
We fix the dimensions to which the input images will be resized before being fed into the model to ensure consistency in the input data and match the input requirements.This is called resizing.

We use a resize method that is bilinear interpolation.
Bilinear Interpolation method performs linear interpolation in both horizontal and vertical directions providing smoother results than nearest neighbor.
This method uses the values of four nearest pixels in the original image to calculate the value of each pixel in the resized image. It performs linear interpolation in one direction and then in another direction.(weighted average)

$$f(x,y)=f(x1,y1)\cdot(1-\alpha)\cdot(1-\beta)+f(x2,y1)\cdot\alpha\cdot(1-\beta)+f(x1,y2)\cdot(1-\alpha)\cdot\beta+f(x2,y2)\cdot\alpha\cdot\beta$$

X1 and x2 are the x coordinates of nearest left and right pixels,
y1 and y2 are the y-coordinates of the nearest top and bottom pixels,
α and β are the fractional distances from the nearest pixels.

$$\alpha = x - x1/x2 - x1$$
$$\beta = y - y1/y2 - y1$$

Interpolate in one dimension:

$$Ix1 = f(x1,y1) \cdot (1-\alpha) + f(x2,y1) \cdot \alpha$$
$$Ix2 = f(x1,y2) \cdot (1-\alpha) + f(x2,y2) \cdot \alpha$$

Interpolate in second dimension:

$$I(x,y) = Ix1 \cdot (1-\beta) + Ix2 \cdot \beta$$

$I(x,y)$ is interpolated pixel value at the point(x,y)
$f(x,y)$ represents the known pixel value at that point.

We create a training dataset,testing dataset with validation split of 0.2 .This generates the datasets having image, class label pairs.
Dataset can be configured for performance using buffered prefetching that can yield data from disk without having I/O blocking.

Normalizing the data:
The RGB channel values are in the [0, 255] range. This is not ideal for a neural network; in general we should make input values small.
This is called normalization (Rescaling). Normalized input values stabilize and accelerate the training process by helping the network to converge. Large input values can lead to large gradients. Exploding gradients will lead to instability . Activation functions(tanh,sigmoid,etc) assume input values to be in a certain range. Normalization ensures that these functions work in the most effective range.
Rescaling the pixel values so that they lie within a confined range.

We use Min-Max Normalization (linear scaling):
This method of normalizing data involves transforming the original data linearly.

$$X=(X-Xmin) / (Xmax-Xmin)$$

Division by constant : Scales pixel values by dividing by a constant value, commonly used to fit data into a desired range.

$$x=x/constant$$

To normalize [0,255] to [0,1] we do:
Image=(Image/255.0)
To normalize [0,255] to [-1.1] we do:
Image=(Image/255.0)*2.0-1.0

In many cases we find that training accuracy increases linearly over time whereas validation accuracy stalls. This is a sign of overfitting that occurs due to a small number of training

examples. The model learns from noises and unwanted details from training examples.The model has a difficult time generalizing to a new dataset. It negatively impacts performance
So we can overcome this by augmentation.
Augmentation:
Augmentation is used to increase the size of the dataset and its variance. If the input set consists of data augmentation like flipping, rotation ,cropping, translation, scaling ,adding noise ,etc.. Then the model learns all these variations.It increases the accuracy of the model.
We apply random transformation on images that yield believable looking images.
We don’t perform augmentation over validation and test dataset.

Geometric transformation:
1.Linear transformation:Linear transformations are simpler and typically involve operations like rotation, scaling, and shearing. These transformations can be represented by matrix operations that preserve lines and parallelism in the image.
2.Affine transformation:Affine transformations are more general than linear transformations and include a combination of translation, rotation, scaling, and shearing.

Rotation-Rotating images by a certain angle.
Translation-Shifting images horizontally or vertically.
Scaling-Zooming in or out.
Flipping-Horizontal or vertical flipping of images.
Shearing-Apply shear transformation to images.

Color space Adjustments
Brightness-Randomly adjust brightness of image.
Saturation-Adjust color saturation.

Cutout:
Cropping a center part of the image.

Model construction(CNN):

Input Layer: Used to accept input images, and dispatched to next layers for feature extraction. This layer may normalize the pixel values of the images from the range [0, 255] to [0, 1].

Feature Extraction:
Feature extraction is a vital step in machine learning that involves transforming raw data into a set of meaningful features that can be used to train classification models.
Deep learning:Deep learning is a subset of machine learning that uses multi layered neural networks called deep neural networks, to simulate the complex decision-making power of the human brain.
Convolution Neural Network(CNN) :Automatically learn the hierarchical features from images.

Here we used CNN algorithm because a Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance to various aspects/objects in the image and be able to differentiate one from the other. Convolutional neural network is composed of multiple building blocks, such as convolution layers, pooling layers, and fully connected layers. CNNs are used for image classification and recognition because of their high accuracy.

1) Convolution Layer:

Used to extract features (feature maps) from input images using image matrix and a filter (Kernel).There are many filters and each of them detects a different feature,for example one detect edges other detects texture,etc..This layer is responsible for extracting features from the input image by sliding a set of filters over it, performing element-wise multiplications, and summing up the results.
a.Initialize a set of filters with random values. Each filter has the same depth as the input data (3D tensor) but with a specified height and width .
b.Each filter(small grid of weights used to extract features) slides over the input data with a specified stride(number of pixels the filter moves over the image).The filters are determined through a training process and the number of feature maps is equal to the number of filters used.No padding is added around the input. The convolution operation is performed only on valid positions where the filter completely overlaps with the input.The output feature map is smaller than the input feature map, as some spatial information is lost at the edges.
c.For each position of the filter, perform element-wise multiplication between the filter and the section of the input data, then sum up the results to produce a single output value for that position.
d.The output of the convolution operation is a set of feature maps, each corresponding to a different filter. These feature maps highlight different features of the input data.

The convolution operation with stride can be mathematically expressed as follows:

$$\text{Output}(i,j) = \sum_{m=0}^{F_h-1} \sum_{n=0}^{F_w-1} I(i \cdot S + m, j \cdot S + n) \cdot K(m,n)$$

Where:

- I is the input image.
- K is the kernel (filter).
- (i,j) are the indices of the output feature map.
- F_h and F_w are the height and width of the kernel, respectively.
- S is the stride.

e.Apply the activation function to each value in the feature maps to introduce non-linearity.

2). ReLU (Rectified-Linear Unit) is the common activation function used. An activation function is applied to the linear combination of inputs. ReLU replaces all negative values in the matrix to zero, which helps faster and effective training. Thus, negative values are not passed to the next layer.

$$\mathrm{ReLU}(x)=\max(0,x)$$

Here x is the input to the activation function.
ReLU is a nonlinear function. Despite being linear for positive inputs, it introduces non-linearity to the model because it outputs zero for negative inputs (negative values indicate that the feature is not present or the filter did not capture the feature).
ReLU is chosen because when the input x is positive, the gradient is 1. This means that during backpropagation, the gradient is directly passed to the previous layer with no attenuation. This property helps in mitigating the vanishing gradient problem that is common with activation functions like sigmoid or tanh, where gradients can become very small and slow down learning.
However when the input x is zero or negative, the ReLU activation outputs zero and its gradient is also zero. This means that during backpropagation, no gradient is passed backward through these neurons. If a neuron remains inactive (i.e., always outputs zero), it doesn’t contribute to learning, leading to the "dying ReLU" problem where neurons effectively become inactive.

Feature Selection and reduction:
1) Pooling: It is used to reduce spatial size of the feature map and thus reduces the parameters required to preserve significant information.
Pooling layers help in extracting dominant features and making the representation more invariant to small distortions.It reduces the amount of computation required for further stages.

Types of pooling layers:
1.Max Pooling.
Takes the maximum value from each patch of the feature map.For example, 2x2 max pooling layer divides the input into 2x2 blocks and selects the maximum value from each block.
It helps retain the most important features while discarding others.
2.Average Pooling.
Takes the average value from each patch of the feature map.It can be used in cases where average is more informative than maximum value.

2)Flatten layer:This layer flattens the 3D feature maps into a 1D vector. This step is required before passing the data to fully connected (dense) layers.It acts as a bridge between the convolutional/pooling layers and the dense layers in a neural network.

The Flatten layer takes a tensor with a shape of (batch_size, height, width, channels) (for 4D tensors) or (batch_size, dim1, dim2) (for 3D tensors) and reshapes it into a tensor of shape (batch_size, height * width * channels) or (batch_size, dim1 * dim2) .The batch dimension (batch_size) remains unchanged during flattening.

In CNNs, after applying convolutional layers, the concept of channels becomes more complex. Each convolutional filter creates a separate feature map that captures different aspects of the input data. The number of channels in the output feature map of a convolutional layer corresponds to the number of filters used in that layer.
In convolutional neural networks (CNNs), the output of a convolutional layer for a single sample (image) might be a 3D tensor representing multiple feature maps and for a batch of images will be a 4D tensor, where each image has its own set of 3D feature maps.

Principal Component Analysis (PCA) is a technique used for feature reduction by projecting high-dimensional data into a lower-dimensional space while retaining the most variance.
CNNs use learned filters for feature extraction, which can capture nonlinear patterns.

Filter-Based Feature Selection involves selecting features based on statistical tests or criteria before applying a model. CNNs use convolutional layers to automatically learn and extract features without explicit statistical filtering.

CNNs employ a form of learned feature extraction and reduction that adapts to the data during training, which is different from the more static, pre-modeling methods like PCA and filter-based selection.

3)Dropout layer(Regularization):
The Dropout layer is a regularization technique used in neural networks to prevent overfitting during training iterations .It randomly drops out (by setting the activation to zero) a number of output units from the layer during the training process. Dropout takes a fractional number as its input value, in the form such as 0.1, 0.2, 0.4, etc. This means dropping out 10%, 20% or 40% of the output units randomly from the applied layer.During inference (when making predictions or evaluating the model), the dropout layer is turned off, meaning no neurons are dropped out. Instead, the output of the neurons is scaled by the dropout rate to maintain the overall network's output scale.

If p is the dropout rate, then the dropout layer randomly sets a fraction p of the activations to zero. During training, the output of the dropout layer is computed as:

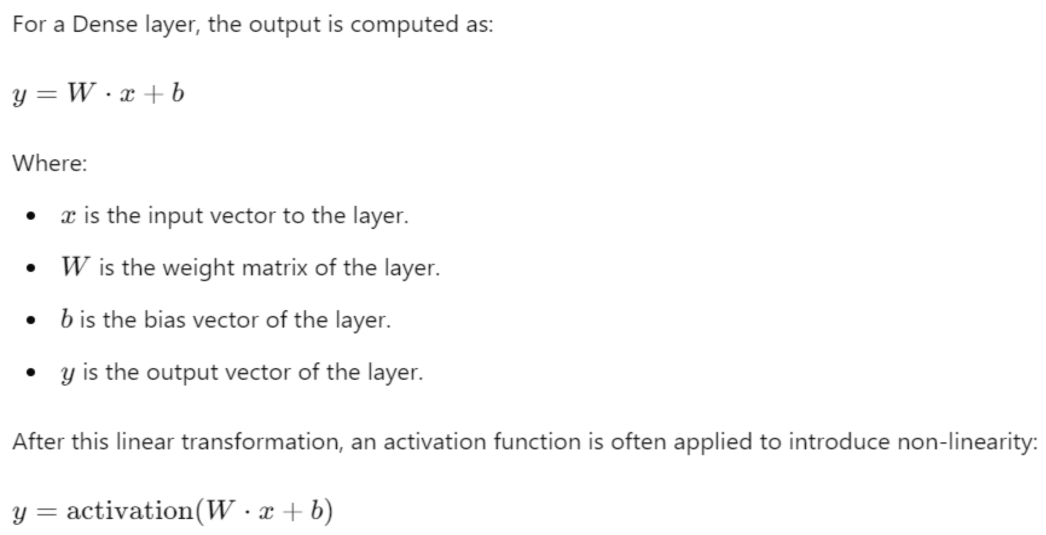
Output=Input×Mask

A binary mask where each element is either 0 (dropped) or 1 (kept). The mask is generated randomly according to the dropout rate p.

During training, to compensate for the fact that some activations are dropped, the remaining activations are scaled up by 1/1-p. This scaling is applied to ensure that the expected value of the activations remains the same as during inference.

Classification:

1) Fully Connected Layer: Dense (fully connected) layers are used to perform classification based on the features extracted by the convolutional and pooling layers. It is used as a hidden layer.
In this layer each neuron is connected to every other neuron in the previous layer. It allows the network to learn complex features.



For a Dense layer, the output is computed as:

$$y = W \cdot x + b$$

Where:

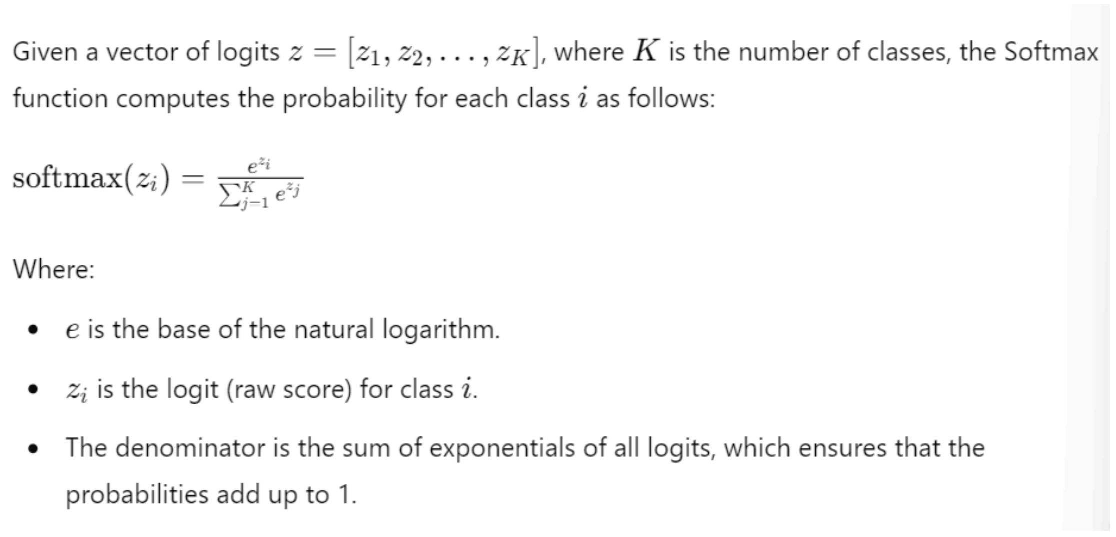
- x is the input vector to the layer.
- W is the weight matrix of the layer.
- b is the bias vector of the layer.
- y is the output vector of the layer.

After this linear transformation, an activation function is often applied to introduce non-linearity:

$$y = \text{activation}(W \cdot x + b)$$

If there are n units in the dense layer each unit corresponds to an output feature.
Each neuron in the dense layer has weight associated with each input feature. If the input vector has n features and the dense layer has m units, the weight matrix has shape (n,m). Each neuron has a bias vector that is of size (m).

2) Softmax Layer: This is the output layer and the final dense layer . It uses softmax activation function .Softmax function converts a vector of raw scores (logits) into probabilities. These probabilities are used to predict the likelihood of each class in a multi-class classification problem. It is used to normalize output of the neural network (between zero and one).



Given a vector of logits $z = [z_1, z_2, \ldots, z_K]$, where K is the number of classes, the Softmax function computes the probability for each class i as follows:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Where:

- e is the base of the natural logarithm.
- z_i is the logit (raw score) for class i.
- The denominator is the sum of exponentials of all logits, which ensures that the probabilities add up to 1.

Softmax uses the exponential function to ensure that higher logits result in higher probabilities. This amplifies the differences between logits, making the predicted class more distinct.

Compiling the model:

Loss function:
Loss function (also known as the objective function or cost function) is a critical component that quantifies how well the model's predictions match the true labels. It measures the difference between the predicted values and the actual values, guiding the optimization process to minimize this difference and improve the model's performance.
Loss function used:
Sparse Categorical cross-entropy:
Used for multi-class classification problems where labels are integers. We use integer labels for our classification over one hot encoded label.

The softmax layer converts vectors of raw scores or logits and outputs a probability(pi) for each class.
The actual class label is represented as integers.(0,1,2,..,N for N classes)

$$\text{Loss} = -\sum_{i=1}^{N} y_i \cdot \log(p_i)$$

where:

- N is the number of classes.
- y_i is a binary indicator (0 or 1) if class label i is the correct classification for the input.
- p_i is the predicted probability of class i (obtained using a softmax activation function).

The optimizer is a crucial component in the training of machine learning models. Its role is to adjust the weights,bias of the model in order to minimize the loss function, thereby improving the model's performance on the task.The model makes predictions by applying the current parameters (weights and biases) to the input data through the network.Gradients of the loss function with respect to each parameter are computed. This involves applying the chain rule of calculus to propagate the error backward through the network.

Adams Optimizer(Adaptive Moment Estimation) is one of the most popular optimization algorithms used in training neural networks. It combines features from two other optimization methods: AdaGrad and RMSProp.It adjusts the model weights during training.

Adam computes individual learning rates for each parameter based on estimates of first and second moments of the gradients (mean and variance). This means that it adapts the learning rate for each parameter separately, which helps in converging faster and more efficiently.

The first moment(m) of the gradient is essentially the mean or moving average of the gradients over time. It helps in capturing the direction of the gradient

The second moment(v) is the moving average of the squared gradients.It measures the variance of the magnitude of the gradients. This moment helps in scaling the learning rates by taking into account the variability of the gradients.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

- m_t is the first moment estimate at time step t.
- β_1 is the decay rate for the first moment (typically around 0.9).
- g_t is the gradient at time step t.

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

- v_t is the second moment estimate at time step t.
- β_2 is the decay rate for the second moment (typically around 0.999).
- g_t^2 is the squared gradient at time step t.

During the initial stages of training, the estimates for the first and second moments are biased towards zero because they are initialized as zero vectors

1. **Bias-Corrected First Moment**:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

   - Here, $\hat{m}_t$ is the bias-corrected first moment estimate at time step t.

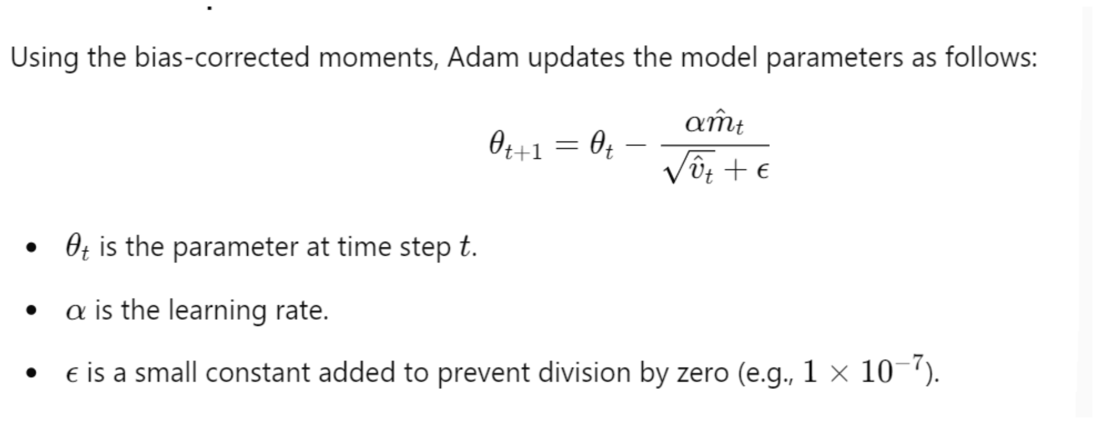
2. **Bias-Corrected Second Moment**:

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

   - Here, $\hat{v}_t$ is the bias-corrected second moment estimate at time step t.

Although Adam adapts learning rates individually for each parameter, it still uses a global learning rate (α) in the update rule.The global learning rate alphaα controls the overall scale of

the updates. It ensures that the magnitude of updates is appropriate for the problem and prevents excessively large or small updates.
By adapting the learning rate for each parameter based on the gradient moments, Adam can handle varying scales of gradients and parameter updates. This adaptability helps in dealing with sparse gradients and irregularities in the gradient landscape.

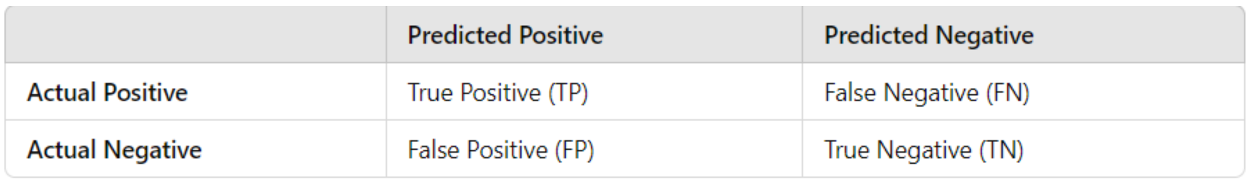

Using the bias-corrected moments, Adam updates the model parameters as follows:

$$\theta_{t+1} = \theta_t - \frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

- θ_t is the parameter at time step t.
- α is the learning rate.
- ϵ is a small constant added to prevent division by zero (e.g., 1×10^{-7}).

Training the model:
We train the model on the training data. We also provide the validation data to evaluate the model during training. Number of epochs represents the number of passes through the entire dataset. Batch size is the number of samples used per gradient update.

Model evaluation:
Evaluate the model on the test set to gauge its performance on unseen data.
Metrics used for evaluation are:
Classification metrics
1.Accuracy: The ratio of the number of correct predictions to the total number of predictions.
2.Precision:The ratio of true positive predictions to the total predicted positives.
3.Recall:The ratio of true positive predictions to the total actual positives.
4.Macro F1 score:The harmonic mean of precision and recall.
For a class i
Accuracy=(TP+TN)/(FP+FN+TP+TN)
Precision=TP/(TP+FP)
Recall=TP/(TP+FN)
Macro F1 Score=2*Precision*Recall/(Precision+Recall)



	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Making predictions:
Predict labels of new samples using the trained model.