**448-HW2**

**Furkan Tuna**

**1. How did you make the input size compatible with the network?**

The images are resized to 299x299 pixels. The InceptionV3 model expects input images of this size along with three color channels (RGB).

1. **Load and Resize Images:**

* Images are loaded from the specified directory.
* Each image is resized to 299x299 pixels using the PIL library.

1. **Normalize Images:**

* Images are normalized to the range [-1, 1] using the formula (images / 127.5) - 1.

1. **Model Input Shape:**

* The InceptionV3 model is instantiated with input\_shape = (299, 299, 3) to ensure it expects images of size 299x299 with 3 channels.

**2. How did you normalize the input?**

I normalized the input images using the formula **(images / 127.5) - 1**. This normalization scales the pixel values from the range [0, 255] to the range [-1, 1], which is the expected input range for the InceptionV3 model.

This normalization ensures that the images are properly scaled before they are fed into the InceptionV3 model. The division by 127.5 converts the pixel values to the range [0, 2], and then subtracting 1 shifts the range to [-1, 1]. This is a common preprocessing step for models that use the InceptionV3 architecture.

**3. What parts of the network architecture did you modify? How did you modify the last layer?**

The InceptionV3 network architecture is modified by adding custom layers on top of the base model. Specifically, we add a global average pooling layer, a dense layer, a dropout layer, and a final dense layer for classification.

1. **Base Model:** The InceptionV3 model is loaded without the top layers (include\_top=False). This means that the fully connected layers at the top of the network (which are used for classification in the original model) are not included.
2. **Global Average Pooling:** This layer aggregates the spatial dimensions of the feature maps, resulting in a single 1x1 feature map per channel. This significantly reduces the number of parameters.
3. **Dense Layer with ReLU:** Adding a dense layer with 1024 units and ReLU activation helps in learning complex patterns from the features extracted by the base model.
4. **Dropout Layer:** Dropout is used to prevent overfitting by randomly setting a fraction of input units to 0 at each update during training time, which helps the network to generalize better.
5. **Final Dense Layer with Softmax:** The final layer has 3 units (corresponding to the 3 classes in your classification problem) and uses the softmax activation function to output probabilities for each class.

This modified architecture builds on the powerful feature extraction capabilities of the pre-trained InceptionV3 model

**4. What loss function did you use in backpropagation?**

The loss function used in the backpropagation process is sparse\_categorical\_crossentropy. This loss function is appropriate for multi-class classification problems where the labels are provided as integers rather than one-hot encoded vectors.

* **Sparse Categorical Cross-entropy:**

This loss function is used for multi-class classification problems.

1. It is similar to categorical\_crossentropy, but it expects the labels to be provided as integers (e.g., 0, 1, 2) rather than one-hot encoded vectors (e.g., [1, 0, 0], [0, 1, 0], [0, 0, 1]).
2. The loss function calculates the cross-entropy loss between the true labels and the predicted probabilities output by the model.

**Why Use sparse\_categorical\_crossentropy?**

1. **Efficiency:**

When the labels are integers, using sparse\_categorical\_crossentropy is more memory efficient because it doesn't require converting the labels to one-hot encoded vectors.

1. **Suitability for Integer Labels:**

Since your labels are integers (as seen from the loading data part), sparse\_categorical\_crossentropy is the appropriate choice.

**5. How did you select the parameters related to backpropagation? For example, did you use any optimizer? If so, what were the parameters of this optimizer and how did you select their**

**values?**

The Adam optimizer is used for backpropagation. The parameters of the Adam optimizer, as well as their selected values, are as follows:

* **Optimizer Selection**
* **Adam (Adaptive Moment Estimation):**

**Why Adam?**

* **Efficiency:** Adam combines the advantages of two other popular optimization algorithms: AdaGrad (adaptive gradient algorithm) and RMSProp (Root Mean Square Propagation). It is computationally efficient and has low memory requirements.
* **Adaptivity:** It adapts the learning rate for each parameter, making it well-suited for problems with sparse gradients.
* **Robustness:** Adam works well across a wide range of deep learning problems, including image classification tasks.

**Parameters of the Adam Optimizer**

* **Learning Rate (learning\_rate):**

Value Used: 0.001

* **Selection Reason:**

The learning rate of 0.001 is a commonly used default value for the Adam optimizer. It is generally effective in finding a good balance between convergence speed and stability.

**6. How did you address the class-imbalance problem?**

I address the class imbalance problem by applying over sampling technique. Oversampling is a data augmentation technique utilized to address class imbalance problems in which one class significantly outnumbers the others. It aims to rebalance training data distribution by amplifying the volume of instances that belong to the under-represented class. I Apply oversampling only to the training data to avoid any bias in the validation set.