**Exercise 3 – Code explanation Resnet and Alexnet**

**Model: Resnet and Alexnet**

**Dataset source:** <https://www.cs.toronto.edu/~kriz/cifar.html>

**Group Members:**

* MICHAEL KURNIAWAN (2602070206)
* BRYAN ORVILLE AUDRIC (2602160750)
* JOSEL LINUS (2602076216)
* VINCENT TANJAYA (2602076916)

**Link Resnet collab:** <https://colab.research.google.com/drive/1I0jL9hspNpfxX2TjJApOI003TgIOHKqp>

**Link Alexnet collab:** <https://colab.research.google.com/drive/18Hg3iud30IWuIPclPfoZd5mSHINyyMv5#scrollTo=bX2pWSX8FrDz>

**Resnet**

1. **Importing Libraries:**

A close-up of a text

Description automatically generated

* **Data loading and Preprocessing:**

1. Datasets: in this library we import pre-built datasets (CIFAR) from tensorflows
2. Numpy(np): to provides efficient methods for working with arrays and matrices, as well as mathematical functions that are useful for manipulating and processing data in machine learning models.
3. CV2: is used for advanced image processing, including reading, writing, and transforming images.
4. Train\_test\_split: this library helps split our dataset into training and testing dataset, making sure our model has separate data for evaluation.

* Model Building:

1. Sequential: Model in keras were used for building deep learning models layer by layer in a straightforward manner.
2. Dense : Fully connected layer for standard neural networks.  
   Flatten : Converts multi-dimensional data into a 1D vector.  
   Dropout: Regularization layer to prevent overfitting by randomly dropping neurons during training.  
   Upsampling2D : Increases the spatial resolution of an image, useful in tasks like image generation or segmentation.  
   Inputlayer : Defines the input shape to the model.  
   batchNormalization : Normalizes inputs to stabilize and speed up training.

* Visualization:

1. Matplotlib (plt): a plotting library that allows to visualize data and results in image.

A close-up of words

Description automatically generated

* In this section we ensure that TensorFlow only uses the necessary GPU memory by enabling memory growth for each GPU, which is particularly useful in multi-GPU systems or when running other tasks on the same GPU.

1. **Dataset**



* In this code we load the CIFAR-10 dataset into 4 variables where X\_train, y\_train for the training set and X\_test, y\_test for the test set

1. **Dataset visualization**

**A screenshot of a computer

Description automatically generated**

* In this code where x\_train.shape, y\_train.shape, x\_test.shape, y\_test.shape it show the shapes (dimensions) of the training and testing data.

A black and white text

Description automatically generated

* In this code we reshape the y\_train and y\_test array from a 2D array to 1D array, making the data easier to work with in most neural network models.

A screen shot of a computer

Description automatically generated

* In this function we deserializes CIFAR-10 binary files and returns them as Python dictionaries, making it possible to access the dataset's image data and labels

A screenshot of a computer code

Description automatically generated

* meta\_data = unpickle('batches.meta') : This loads the metadata from the file 'batches.meta' using the unpickle function defined earlier.
* classes = meta\_data[b'label\_names']: This line retrieves the list of class label names from the meta\_data dictionary.
* classes = [item.decode() for item in classes]: Converts the list of class names from byte strings to regular Python strings.

A white background with black text

Description automatically generated

* visualize the list of class names from the dictionary and decodes the byte strings into regular Python strings, resulting in a list of the 10 class labels for CIFAR-10 such as ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

A screenshot of a computer

Description automatically generated

* This function is to show image from the CIFAR-10 dataset along with its corresponding class label.

1. **Preprocessing**

**A screenshot of a computer code

Description automatically generated**

* In this section we are splitting the data, but there were 50000 training data and 10000 testing data the ratio is 83.33% for training data and 16.67% for testing data The training data will be further split into training data and validation data, where the data split would be:

Training data = 40000 (66.67%)

Validation data = 10000 (16.67%)

Testing data = 10000 (16.67%)

**A screenshot of a computer program

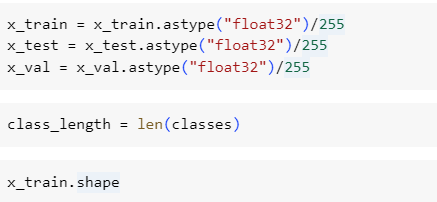
Description automatically generated**

* In this code defines an instance of ImageDataGenerator from the tensorflow.keras.preprocessing.image module, which is used to apply various data augmentation techniques to images during model training. Data augmentation artificially increases the diversity of the training data by applying random transformations, helping prevent overfitting and making the model more robust.
* The parameter explanation:

1. rotation\_range : we Randomly rotates images by up to 20 degrees in both clockwise and counterclockwise directions. It makes the model less sensitive to the orientation of objects in the images.
2. width\_shift\_range : Randomly shifts the image horizontally (left or right) by up to 14% of the image width. Helps the model learn to detect objects that may be off-center or shifted slightly horizontally.
3. height\_shift\_range : Randomly shifts the image vertically (up or down) by up to 14% of the image height. this helps the model detect objects that may be shifted vertically.
4. horizontal\_flip : Randomly flips the image horizontally (left to right). This augments the dataset by flipping images, useful for objects that are symmetrical or where the direction does not matter
5. zoom\_range : Randomly zooms in or out by up to 11% of the image size. Thhis will teaches the model to recognize objects at different scales or distances.
6. brightness\_range : Randomly adjusts the brightness of the image within the range specified (0.9 to 1.1). It helps the model learn to detect objects under different lighting conditions.
7. shear\_range : Applies a shear transformation, which skews the image along the axis, by up to 12%. Distorts the image slightly, helping the model become invariant to skewed or distorted inputs.
8. channel\_shift\_range : Randomly shifts the color channels (R, G, B) by up to 10%. Helps the model handle small variations in color or lighting, which can occur due to environmental conditions.



* In this code we augment the training data



* In this code we normalize the data image to range 0-1 by dividing it with 255

1. **Build the model**

**A screenshot of a computer program

Description automatically generated**

* **Resnet\_model:**

1. tf.keras.applications.ResNet50(...): This function loads the ResNet50 model pre-trained on the ImageNet dataset.
2. include\_top=False: Excludes the final fully connected layers (classification head) of ResNet50, allowing you to add your own layers for classification.
3. weights="imagenet": Initializes the model with weights trained on the ImageNet dataset, allowing for better feature extraction from images.
4. input\_shape=(64, 64, 3): Sets the input shape of the model to 64x64 pixels with 3 color channels (RGB).
5. pooling="avg": Applies average pooling to the output of the ResNet50 layers. This means the output feature maps are averaged into a single feature vector, which reduces dimensionality.

A screen shot of a computer code

Description automatically generated

* **Model :**

1. Sequential: Initializes a sequential model where layers are stacked linearly.
2. InputLayer(input\_shape=(32, 32, 3)): Defines the input shape of the model as 32x32 pixels with 3 colour channels.
3. UpSampling2D((2,2)): Upsamples the input image by a factor of 2 in both dimensions, changing the input size from (32, 32, 3) to (64, 64, 3) before passing it to the ResNet model. This is important because ResNet50 expects inputs of at least 64x64 pixels.
4. resnet\_model: Incorporates the pre-trained ResNet50 model.
5. Flatten(): Flattens the 3D output from the previous layer into a 1D array, preparing it for the fully connected layers.
6. BatchNormalization(): Normalizes the output of the previous layer, which can help accelerate training and improve stability.
7. Dense(128, activation='relu'): Adds a fully connected layer with 128 units and ReLU activation function, introducing non-linearity.
8. Dropout(0.25): Applies dropout with a rate of 25%, randomly setting 25% of the input units to 0 during training to prevent overfitting.
9. BatchNormalization: Normalizes the output from the first dense layer.
10. Dense(64, activation='relu'): Adds another fully connected layer with 64 units and ReLU activation.
11. Dropout(0.25): Another dropout layer for regularization.
12. BatchNormalization: Normalizes the output from the second dense layer.
13. Dense(class\_length, activation='softmax'): The final layer outputs predictions for class\_length classes using the softmax activation function, which is suitable for multi-class classification problems.

* Model.summary() : Displays a summary of the model architecture, including the number of layers, output shapes, and the number of parameters at each layer.

A close-up of text

Description automatically generated

* This section enables mixed precision training, which uses both 16-bit (float16) and 32-bit (float32) floating-point types in the model. This can significantly speed up training on compatible hardware (like NVIDIA GPUs) and reduce memory usage without sacrificing model accuracy.



* This callback monitors the validation loss during training and stops training if it doesn't improve for a specified number of epochs, preventing overfitting.



* This line configures the model for training by specifying the optimizer, loss function, and evaluation metrics.



* The fit method is used to train the model on the given training data. It returns a history object, which contains the training and validation metrics for each epoch, allowing you to analyze the model’s performance over time.

A screenshot of a computer

Description automatically generated

* model.save("ResNet50.h5"): This line saves the trained model to a file named ResNet50.h5. This allows you to reuse the model later without retraining it.
* load\_model("ResNet50.h5"): This line loads the previously saved model from the file. This is useful when you want to evaluate the model or make predictions without retraining it.

A computer screen shot of a computer code

Description automatically generated



* loss: 0.4128: The model's loss on the test set is 0.4128. A lower loss indicates that the model's predictions are closer to the true labels, but the interpretation of this value also depends on the context of the problem and the specific loss function used.
* accuracy: 0.8891: The model achieved an accuracy of approximately 88.91% on the test dataset, meaning that it correctly predicted the class labels for about 88.91% of the samples.
* but with a loss of 0.41 there may still some errors or inefficiencies the model could improve on.

1. **Visualize the results**

A computer screen shot of a code

Description automatically generated

* This code creates a plot of the training and validation accuracy over epochs.
* The x-axis represents the number of epochs, and the y-axis shows the accuracy.

A graph of a curve

Description automatically generated with medium confidence

**Accuracy vs Epochs:**

* **X-axis (Epochs):** Number of epochs (training cycles).
* **Y-axis (Accuracy):** Measures the percentage of correct predictions.
* **Training accuracy (blue line):** The model starts with lower accuracy, but steadily improves as the epochs progress, reaching above 90%.
* **Validation accuracy (orange line):** Starts higher than the training accuracy, increases rapidly in the first few epochs, then plateaus at around 90%.

The validation accuracy plateaus while the training accuracy continues to improve, indicating potential overfitting again. The model is learning to predict the training data very well but does not show the same level of improvement on validation data after a certain point.

A computer code with text

Description automatically generated

* This code creates a plot of the training and validation loss over epochs.

A graph of a line

Description automatically generated with medium confidence

**Loss vs Epochs:**

* **X-axis (Epochs)**: Represents the number of training cycles the model has completed.
* **Y-axis (Loss)**: This is the loss function's value, which measures how well or poorly the model is performing.
* **Training loss (blue line)**: Initially, it is quite high, indicating that the model's predictions are inaccurate. As training progresses, the loss decreases steadily, showing that the model is learning and making better predictions.
* **Validation loss (orange line)**: Starts lower than the training loss and decreases initially, but then plateaus or slightly increases after a few epochs. This suggests that while the model is performing better on the training data, its performance on unseen (validation) data stops improving and slightly worsens towards the end.

The trend shows that after a certain point (around epoch 7), the model starts overfitting. While the training loss keeps improving, the validation loss remains stable/not decreasing, indicating the model is struggling to increase its accuracy.

**Overall Conclusion:**

This project successfully implements Resnet model using dataset CIFAR10. With the step-by-step workflow above by using Resnet we get accuracy over 88.91% and loss around 0.4128. So, we can conclude that the model is fairly well and is reasonably accurate in classifying images from the CIFAR-10 dataset.

**Alexnet**

1. Importing Libraries

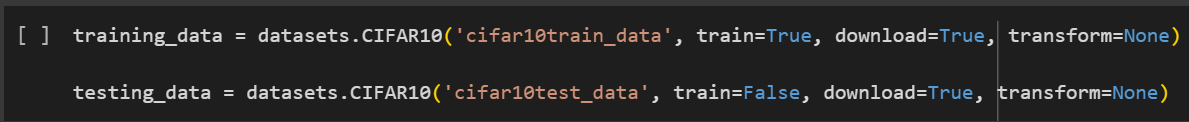
The following libraries are imported:

A screenshot of a computer

Description automatically generated

* PyTorch: Used for building and training deep learning models, also helpful for data preprocessing.
* Torchvision: A package in PyTorch specifically for image processing tasks. It includes tools for loading the CIFAR-10 dataset and applying image transformations.
* NumPy: Provides efficient matrix operations and data manipulations, which are useful for handling datasets.
* Matplotlib: For visualizing data and model performance during training.

1. Data loading, (Visualizations are not explained since the processes are similar to the resnet version), and Preprocessing.



Dataset: CIFAR-10, a commonly used image dataset consisting of 60,000 32x32 color images in 10 classes, with 50,000 for training and 10,000 for testing.

Data Augmentation: Applied through the torchvision.transforms module, which includes normalization, random cropping, horizontal flipping, etc., to increase the robustness of the model. The parameter of augmentation are the same exact with the resnet version.

A computer screen shot of a program code

Description automatically generated

The training data is also normalized with a mean and standard deviation for each channel, which helps in faster convergence during training.

However, testing and validation data does not receive data augmentation process

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

1. Model Building

The model architecture follows the AlexNet structure:

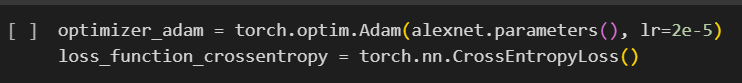
A screenshot of a computer program

Description automatically generated

* Input Layer: Takes 224x224 RGB images (3 channels).
* Convolutional Layers: Multiple convolutional layers that extract features using filters. AlexNet uses ReLU activation functions after each layer.
* Pooling Layers: Max pooling layers downsample the feature maps, reducing the spatial dimensions while preserving important information.
* Fully Connected Layers: At the end of the convolutional layers, the features are flattened and passed through fully connected layers.
* Dropout Layers: Applied after certain fully connected layers to reduce overfitting by randomly setting some outputs to zero.
* Output Layer: the last layer of Alexnet is adjusted to output number of classes in the cifar10 dataset.



* The torch.optim library is used to define the optimizer (e.g., Adam or SGD) and the loss function is the cross-entropy loss, which is suitable for multi-class classification tasks.



1. Training the Model

The training loop in the notebook (trainModel() function) performs the following steps:

Forward Propagation: Passes the input data through the AlexNet model.

Loss Calculation: Computes the difference between the predicted and true labels using cross-entropy loss.

Backpropagation: Updates the model weights using the optimizer based on the computed gradients.

Evaluation: At the end of each epoch, the model's performance is evaluated on the test dataset.

The number of epochs and the batch size are defined in the script to train the model progressively, allowing it to learn complex patterns from the dataset.

1. Performance and Accuracy

Accuracy during training

A graph with blue and orange lines

Description automatically generated

Loss during training

A graph of loss and validation

Description automatically generated

The curve of both graphs is quite like the resnet model. This indicates a slight overfit during the later epoch, causing the model to struggle when it attempts to further increase its accuracy.

After training, the AlexNet model achieves an accuracy of 89.32% and a loss of 0.3274 on the CIFAR-10 test dataset. This accuracy is measured and printed at the end of each epoch.

A screenshot of a computer program

Description automatically generated

Conclusion

This AlexNet model effectively learns to classify the CIFAR-10 dataset, achieving a final test accuracy of approximately 89.32%. Despite some room for improvement, the model demonstrates AlexNet's capacity for image recognition tasks. Further tuning of hyperparameters or the addition of data augmentation techniques could improve its generalization.