DM Final Project

Rikshith Tirumanpuri
U45426485

What is the problem we are trying to solve ?

The challenge involves a classification task employing the CIFAR-100 dataset, comprising 32x32 colour images distributed across 100 distinct classes. The primary objective is to develop a neural network capable of effectively categorising these images into their respective classes.

The code, derived from Daniel Sawyer's implementation, adopts a functional organisation for a distinct structure. The dataset undergoes division into training, validation, and test sets, with 10% of the data reserved for validation purposes. The initial accuracy stands at 44%, prompting the goal of enhancing this performance.

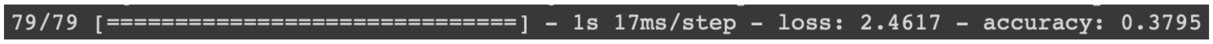
Improvements can be pursued through alterations to the neural network architecture, training parameters, or data preprocessing. While an increase in the number of epochs is permissible, it must not be the sole modification. The objective is to showcase improved performance within 15 epochs or present a well-founded argument if such an outcome is unattainable.

Furthermore, the task includes reporting the best and worst class accuracies achieved, along with details on the dataset's example count.

In essence, the problem revolves around augmenting classification accuracy on the CIFAR-100 dataset via neural networks, exploring diverse modifications and elucidating the rationale behind each change made.

What is the goal of the project and the baseline accuracy ?

By running the default model, I got a baseline accuracy of 37.95%.



```
79/79 [==============================] - 1s 17ms/step - loss: 2.4617 - accuracy: 0.3795
```

The goal is to enhance this accuracy through adjustments to the architecture, training parameters, or data preprocessing, with the expectation of achieving better performance at 15 epochs or providing a reasoned explanation if it's not feasible.

Model Changes or Improvements:

Normalize the data.

- The first thing to do when training neural networks is to normalize the data so that the mean is 0 and the variance is 1. This results in a stable training.
- This didn't help improve the accuracy.

```
def normalize_img(image, label):
  """Normalizes images: `uint8` -> `float32`."""
  return tf.cast(image, tf.float32) / 255., label
```

```
ds_train = ds_train.map(normalize_img,
num_parallel_calls=tf.data.experimental.AUTOTUNE)
dsvalid = dsvalid.map(normalize_img, num_parallel_calls=tf.data.experimental.AUTOTUNE)
 ds_test = ds_test.map(normalize_img, num_parallel_calls=tf.data.experimental.AUTOTUNE)
```

Adding more layers

- Stacking more layers (third convolution and batch normalization) didn't help.

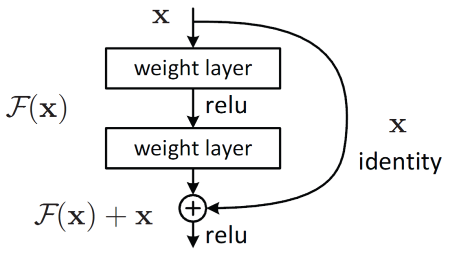
```
ly = tfk_layers.Conv2D(64, 3)(ly)
ly = tfk_layers.BatchNormalization()(ly)
ly = tfk_layers.Activation('relu')(ly)
```

- At this point the accuracy was **42.89** on the test dataset.

Increasing the channel depth

- To prevent underfitting it is a good idea to add more channels.
- Adding channels only didn't help. However, by adding one more layer from the last step and increasing the channel dimension by 2, improved the accuracy to **45.17** on the test dataset.

Adding residual / skip a connection



x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

- Skip connection helps training deeper network
- Also, the skip connections enable learning identity function which makes the network pick just enough parameters.
- Adding a skip connection to the previous setup improved the accuracy to **47.61.**

```
def residual_block(x, filters, kernel_size=3):
  # Shortcut
  shortcut = x

  # First layer
  x = tfk_layers.Conv2D(filters, kernel_size, padding='same')(x)
  x = tfk_layers.BatchNormalization()(x)
  x = tfk_layers.Activation('relu')(x)

  # Second layer
  x = tfk_layers.Conv2D(filters, kernel_size, padding='same')(x)
  x = tfk_layers.BatchNormalization()(x)

  # Adding the shortcut to the output of the block
  x = tfk_layers.Add()([shortcut, x])
  x = tfk_layers.Activation('relu')(x)

  return x
```

What Factors worked to improve the overall accuracy ?

The incorporation of a skip connection and an increase in network depth contributed to an improved accuracy compared to the baseline, demonstrating the effectiveness of these architectural enhancements.

What Factors did not work to improve the overall accuracy?

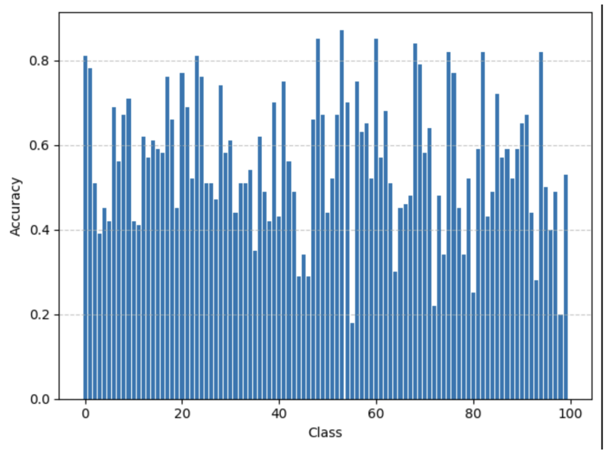
Switching the activation function to ELU (Exponential Linear Unit) resulted in a reduction in test accuracy, indicating that the choice of activation function had a negative impact on the model's overall performance.

How does implementation of ensembles contribute to overall increase in accuracy?

Ensemble learning in CNNs combines predictions from multiple models, leveraging their diversity to enhance overall performance and improve generalization. This is achieved through techniques like voting or averaging.

- I ran 5 different models for 15 epochs, and ensemble the models in two different ways: summation and mode.
- In *summation,* the logits from the 5 models are summed and the classes with the highest logit value is the predicted class.
- In *mode*, the class the majority of the 5 models agree is chosen as the predicted class.
- Ensemble using *summation* improves the accuracy to **57.12**, and
- ensemble using *mode* improves the accuracy to **55.14**.

Which ones are the best and worst performing classes ?



D

The bar plot shows the accuracy on the y-axis and the class number on the x-axis.

Class 55 is the worst performing class with accuracy **18.0**, and class 53 is the best performing class with accuracy **87.0**.