Assignment 1

Department of Electrical and Computer Engineering
North South University

CSE465

Sec: 03

Submitted by

Anik Roy Pranto

192 1219 042

Topic:

Comparative Analysis of Custom CNN and Feedforward Network for Tiny ImageNet Classification.

1. **Project Summary:**

This project aims to compare the effectiveness of a custom CNN (Convolutional Neural Network) and a Feedforward Network for image classification using a small dataset from Tiny ImageNet. The dataset comprises 500-1000 images across ten distinct categories. The objective is to understand which neural network architecture performs better in scenarios with limited data.

Our primary goal is to conduct a comparative study between a custom CNN and a Feedforward Network in the context of image classification using a limited dataset from Tiny ImageNet. Firstly, we will curate a dataset consisting of 500-1000 images from Tiny ImageNet, evenly distributed across ten distinct categories. Then we will design and implement a custom CNN architecture and a Feedforward Network with a suitable structure. To ensure optimal performance, the models will be fine-tuned using regularization technique and validated to prevent overfitting. Subsequently, a crucial phase involves testing the models on entirely new images that were not part of the training set. The objective is to assess their classification accuracy and efficiency. Through a detailed analysis of the results, we will gain insights into the comparative effectiveness of these two models when dealing with a limited dataset. This study contributes to a better understanding of their respective capabilities in image classification tasks with constrained data availability.

Convolutional Neural Network (CNN) outperforms the Feedforward Neural Network (FFN) in both test accuracy and test loss. CNN achieves a higher accuracy (76.00% compared to 51.00% for FFN) and a lower test loss (1.0195 compared to 1.5004 for FFN). By comparing the accuracy results of the custom CNN and the Feedforward Network, we aim to draw conclusions that the CNN is better suited for tasks related to image processing and pattern recognition.

2. **Dataset:**

For tiny image net dataset, we used Stanford library for data collection. We select 1000 tiny images with 10 different classes and split them into 800 training and 100 for testing and 100 validations.

Dataset Link:

http://cs231n.stanford.edu/tiny-imagenet-200.zip

3. **Hardware usage:**

We used **Google Colab TPU** as GPU to collaborate remotely to train our dataset.

Model Name	**Required time to train**	**Average time spent on each epoch**
CNN Model	19.03 minutes	3.65 seconds
Feed Forward Network	8.14 minutes	1.45 seconds

Google TPU (Tensor Processing Unit) is an application-specific integrated circuit (ASIC) developed by Google for neural network machine learning using TensorFlow software. It is designed for high-volume, low-precision computation with specialized features like the Matrix Multiply Unit (MXU) and proprietary interconnect topology. Google TPU includes Sparse Cores for accelerating models relying on embeddings while being efficient in die area and power usage. Therefore, the hardware usage of Google TPU during training is optimized for performance, efficiency, and scalability.

4. **The number of parameters and number of computations for a forward pass of two models:**

Two models we used.

1. CNN and

2. Feed Forward Network.

CNN:

CNN MODEL

```
model = tf.keras.Sequential(
    [
     tf.keras.layers.Rescaling(1./255),
     tf.keras.layers.Conv2D(50, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu', input_shape=(64, 64, 3)),
     tf.keras.layers.Conv2D(75, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'),
     tf.keras.layers.MaxPool2D(pool_size=(2,2)),
     tf.keras.layers.Dropout(0.25),
     tf.keras.layers.Conv2D(125, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'),
     tf.keras.layers.MaxPool2D(pool_size=(2,2)),
     tf.keras.layers.Dropout(0.25),
     tf.keras.layers.Flatten(),
     tf.keras.layers.Dense(500, activation='relu'),
     tf.keras.layers.Dropout(0.4),
     tf.keras.layers.Dense(250, activation='relu'),
     tf.keras.layers.Dropout(0.2),
     tf.keras.layers.Dense(10,activation="softmax"),
    ]
)
```

```
[ ] model.compile(
        optimizer="adam",
        loss=tf.losses.SparseCategoricalCrossentropy(from_logits = False),
        metrics=['accuracy']
    )
```

According to our used model for a forward pass for using TensorFlow we got these calculations for CNN.

```
In [27]: model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 64, 64, 3)         0

 conv2d (Conv2D)             (None, 64, 64, 50)        1400

 conv2d_1 (Conv2D)           (None, 64, 64, 75)        33825

 max_pooling2d (MaxPooling2D  (None, 32, 32, 75)       0
 )

 dropout (Dropout)           (None, 32, 32, 75)        0

 conv2d_2 (Conv2D)           (None, 32, 32, 125)       84500

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 125)      0
 2D)

 dropout_1 (Dropout)         (None, 16, 16, 125)       0

 flatten (Flatten)           (None, 32000)             0

 dense (Dense)               (None, 500)               16000500

 dropout_2 (Dropout)         (None, 500)               0

 dense_1 (Dense)             (None, 250)               125250

 dropout_3 (Dropout)         (None, 250)               0

 dense_2 (Dense)             (None, 10)                2510

=================================================================
Total params: 16,247,985
Trainable params: 16,247,985
Non-trainable params: 0
_________________________________________________________________
```

Step by Step Calculations:

1. **Convolutional Layer 1:**

   Input shape (64,64,3) 50 filters with kernel size of 3*3*3

   Number of Parameters:

   50 (number of filters) * (3 * 3 * 3) (size of each filter) + 50 (biases) = 1400 parameters

Number of Computations:

Stride = 1 and same padding means the output has the same dimension as input

(64 *64*50) * **(**3 * 3 * 3) = 5529600

2. Convolutional Layer 2:

75 filters with filter 3*3 and input size = 64 * 64 * 50

Number of Parameters:

75 * (3 * 3 * 50) (size of each filter) + 75 (biases) = 33825 parameters

Number of Computations:

The number of computations is equal to the number of parameters multiplied by the number of output pixels, which is.

(64 * 64 *75) * (3*3*50) = 138240000

3. Max Pooling Layer 1:

pool size 2x2 and input shape 64x64x7.The number of parameters is 0 and

Number of Computations:

75 * 2 * 2 * 32 * 32 = 307200

4. Dropout Layer 1:

Dropout layers have no parameters, and they don't introduce additional computations.

5. Convolutional Layer 3:

125 filters of size 3x3 and input shape 32x32x75.

Number of Parameters:

125 * (3 * 3 * 75) (size of each filter) + 125 (biases) = 84500 parameters

Number of Computations:

75 * 3 * 3 * 125 * 32 * 32 = 86400000

6. Max Pooling Layer 2:

Pool size 2x2 and input shape 32x32x125: The number of parameters is 0.

The number of computations is 125 * 2 * 2 * 16 * 16 = 128000

7. Dropout Layer 2:

Dropout layers have no parameters, and they don't introduce additional computations.

8. Flatten Layer:

This layer simply reshapes the data and doesn't introduce additional parameters or computations.

9. Dense Layer 1:

With 500 units and input shape 32000.

Number of Parameters:

32000 * 500 + 500 (bias) = 16000500.

Number of Computations:

32000 * 500 = 16000000.

10. Dropout Layer 3:

Dropout layers have no parameters, and they don't introduce additional computations.

11. Dense Layer 2:

With 250 units and input shape 500

Number of Parameters:

250 * 500 + 250 (biases) = 125250 parameters

Number of Computations:

500 * 250 = 125000

12. Dropout Layer 4:

Dropout layers have no parameters, and they don't introduce additional computations.

13. Dense Layer 3 (Output Layer):

With 10 units and input shape 250

Number of Parameters:

10 * 250 (input size) + 10 (biases) = 2510 parameters

Number of Computations:

The number of computations is 250 * 10 = 2500.

Total number of Parameters = 1400 + 33825 + 84500 + 16000500 + 125250 + 2510 = 162479855 parameters

The total number of computations = 5529600 **+** 138240000 + 307200 + 86400000 + 128000 + 16000000 + 125000 + 2500 = 122332300.

Feed Forward Network:

Feed-Forward MODEL

```
model_ff = tf.keras.Sequential(
    [
     tf.keras.layers.Flatten(input_shape=(64,64,3)),
     tf.keras.layers.Dense(4096, activation='relu'),
     tf.keras.layers.Dense(2048, activation='relu'),
     tf.keras.layers.Dense(1024, activation='relu'),
     tf.keras.layers.Dense(512, activation='relu'),
     tf.keras.layers.Dense(256, activation='relu'),
     tf.keras.layers.Dense(10, activation='softmax')
    ]
)
```

```
model_ff.compile(
    optimizer="adam",
    loss=tf.losses.SparseCategoricalCrossentropy(from_logits = False),
    metrics=['accuracy']
)
```

```
model_ff.fit(
    train_ds,
    validation_data = val_ds,
    epochs = 20
)
```

According to our used model for a forward pass for using TensorFlow we got these calculations for FFN:

```
In [29]: model_ff.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 12288)             0

 dense_3 (Dense)             (None, 4096)              50335744

 dense_4 (Dense)             (None, 2048)              8390656

 dense_5 (Dense)             (None, 1024)              2098176

 dense_6 (Dense)             (None, 512)               524800

 dense_7 (Dense)             (None, 256)               131328

 dense_8 (Dense)             (None, 10)                2570

=================================================================
Total params: 61,483,274
Trainable params: 61,483,274
Non-trainable params: 0
_________________________________________________________________
```

Step by Step Calculations:

1. Input layer (Flatten layer):

There are no parameters in the Flatten layer since it only reshapes the input. The number of computations is zero for this layer.

2. Dense Layer 1:

4096 neurons

Number of Parameters:

(12288 input units + 1 bias) * 4096 neurons = 50335744 parameters.

Number of Computations:

Total computations = 12890 * 4096 neurons = 52699136 computations.

3. Dense Layer 2:

2048 neurons

Number of Parameters:

(4096 + 1) * 2048 = 8390656 parameters.

Number of Computations:

Total computations = 12890 * 2048 neurons = 26406400 computations.

4. Dense Layer 3:

1024 neurons

Number of Parameters:

(2048 + 1) * 1024 = 2098,176 parameters.

Number of Computations:

Total computations = 12890 * 1024 neurons = 13190400 computations

5. Dense Layer 4:

512 neurons

Number of Parameters:

(1024 + 1) * 512 = 524800 parameters.

Number of Computations:

Total computations = 12890 * 512 neurons = 6599680 computations.

6. Dense Layer 5:

256 neurons

Number of Parameters:

(512 + 1) * 256 = 131328 parameters.

Number of Computations:

Total computations = 12890*256 neurons = 3,300640 computations.

7. Output layer (Dense, 10 neurons with SoftMax activation):

Number of Parameters:

(256 + 1) * 10 = 2570 parameters.

Number of Computations:

Total computations = 258 * 10 neurons = 2580 computations.

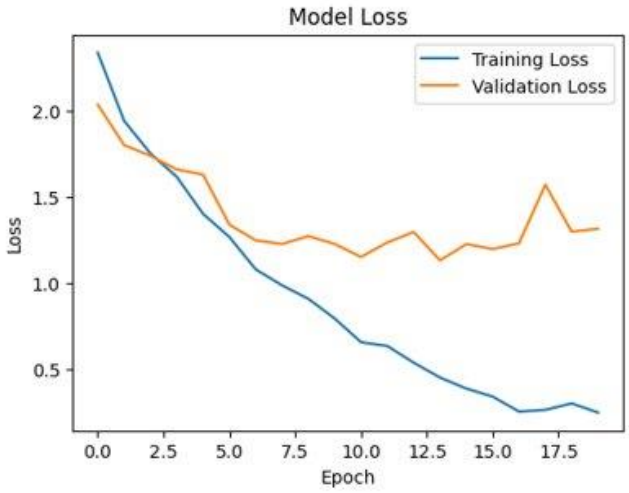
Total number of Parameters = 50335744 + 8390656 + 2098176 + 524800 + 131328 + 2570 = 61483274 parameters.

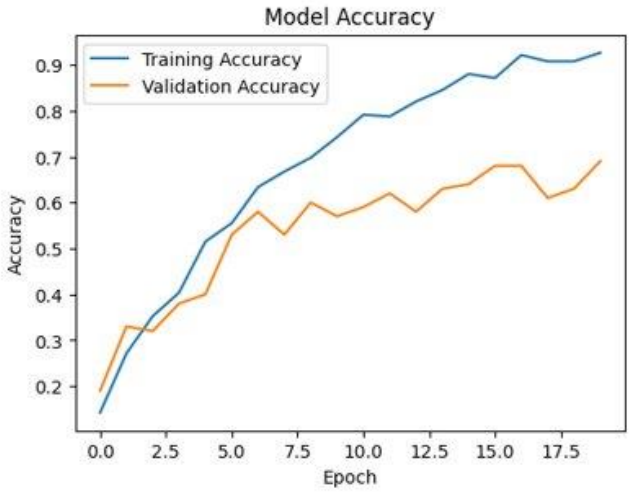
The total number of computations = 52699136 + 26406400 + 13190400 + 6599680 + 3300640 + 2580 = 102198836 computations.

5. Metrics or figures:

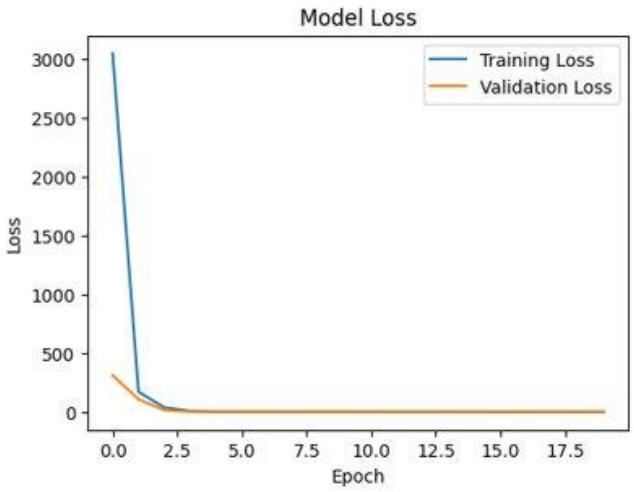
The Training curve (learning and validation)

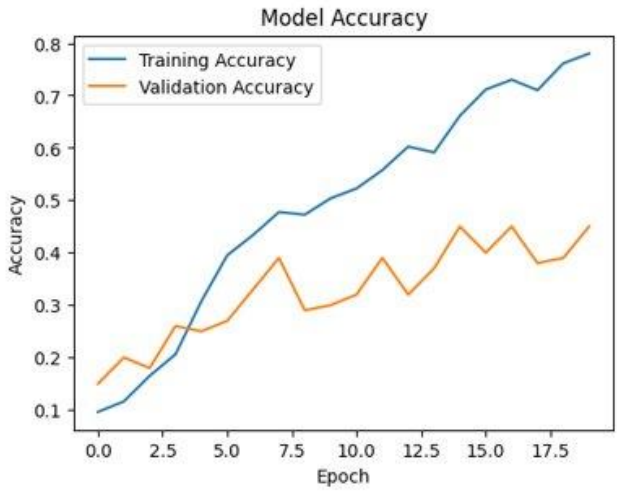
CNN Model:





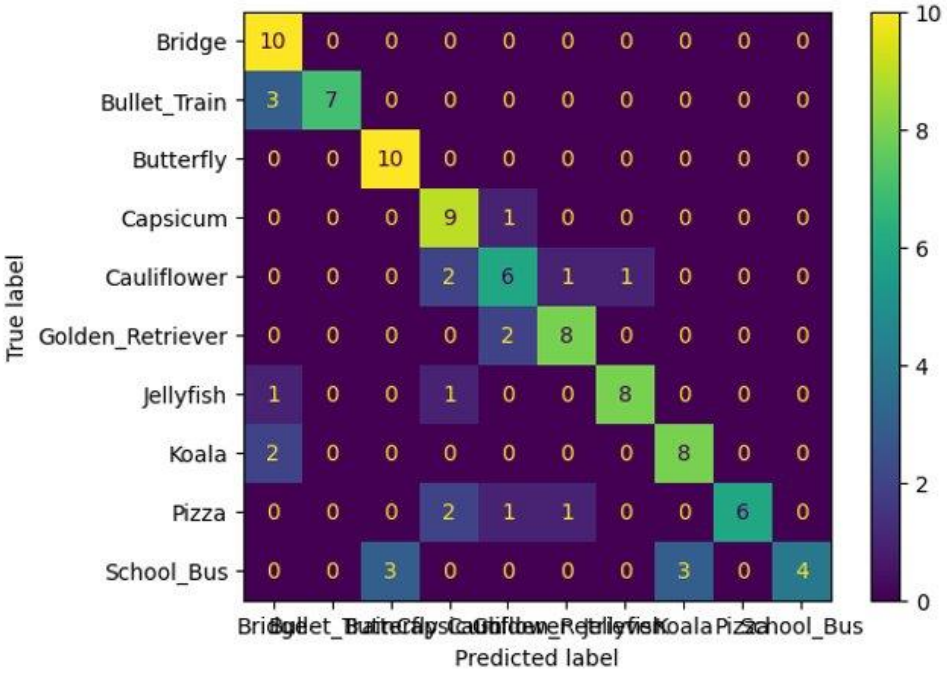
FFN Model:





Confusion Matrix (CNN):

CNN Model:



FFN Model: