ECE472 – Project 4

Jonathan Lam

October 1, 2020

Project description: Training a (ResNet) CNN on CIFAR-10, CIFAR-100. CIFAR-10 accuracy should be state-of-the-art, and CIFAR-100 top-5 accuracy should be at least 80%.

Contents

1 Model

1.1 Dataset

CIFAR datasets were the Python datasets downloaded from [1]. Each dataset was already split into 50000/10000 train/test. Images are 32x32 color images, and the (categorical) labels are 0-9 for CIFAR-10 and 0-99 ("fine labels") for CIFAR-100. The samples are equally split between the different categories.

1.2 Image preprocessing

The pixel values were manually standardized to a $N(0, 1)$ distribution, and then augmented using `tf.keras.preprocesssing.image.ImageDataGenerator`. This involves slight shifting, vertical and horizontal flipping, and some angle rotation. See the source code for more details.

1.3 Structure

I used the structure of ResNet-34, pictured in Figure 1, as rough guidance for what an overall network structure should look like. In the end, the number of filters per layer and the number of layers was varied to try to decrease training time and increase accuracy.

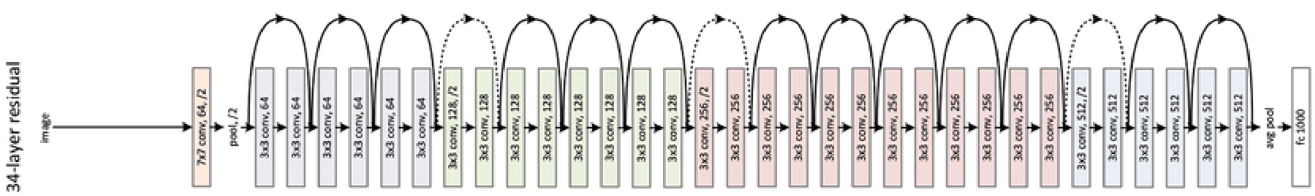

Figure 1: ResNet-34 overall structure. Source: [4]

1.4 ResNet blocks

The individual blocks were the improved ResNet units described in [3]. Namely, these were the "pre-activation" ResNet blocks proposed in that paper, pictured in Figure 2.

In my model, ELUs were used in the place of ReLUs, similarly to the last project. There was also a dropout layer at the end of every ResNet unit for regularization. The `he-normal` initialization method was used for convolutional layer weights.

1.5 Regularization

A small dropout regularization was performed in each ResNet block. This doesn't show up in the ResNet papers [2, 3], but I wanted to try using it since we covered it in class.

Similar to the MNIST classification project, L2 regularization was performed on the weights (this time for the filters on the convolutional layers).

Since the accuracy (CIFAR-10) and top-5 accuracy (CIFAR-100) were similar between the training and test datasets, I believe that this level of regularization is sufficient. In this particular training

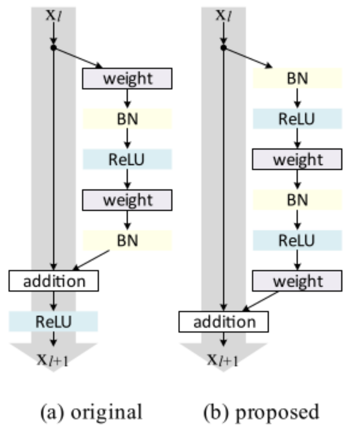

Figure 2: ResNet block structure. (a) The original structure proposed in the original ResNet formulation [2]; (b) The improved structure proposed in [3]. Source: [3]

case for CIFAR-10, the test dataset accuracy is slightly higher than the training dataset accuracy (93.42% on the test dataset as opposed to 90.22% on the training dataset), which is just due to chance.

1.6 Hyperparameter selection/tuning

Hyperparameters were selected manually (see Figure 3) and was not tuned systematically by a builtin tuner like `kerastuner` on a validation set. The reasoning for this is given in the following section (i.e., time constraints). With more time performance could probably be improved further with hyperparameter tuning.

1.7 Differences between CIFAR-10 and CIFAR-100 models

The model used was between the two models was the same, except that the final dense layer had different widths due to the nature of softmax (10 for CIFAR-10, and 100 for CIFAR-100). The only differences were in the data entry (i.e., different filenames, and for CIFAR-100 we were looking at the "fine labels" field rather than the "labels" field of the input) and in the model evaluation: for CIFAR-10, the performance metric was classification accuracy; for CIFAR-100, the performance metric was top-5 classification accuracy.

2 Notes on implementation and training

- Most of the training was performed on Google Colab. Initially (and for all of the previous projects), I had been running the Python code on my desktop computer (i7-2600, no TensorFlow-compatible GPU) and laptop (i7-7500U, no TensorFlow-compatible GPU), both

Hyperparameter	Selected value	Justification
L2 coefficient	0.0001	The default value for `tf.keras.regularizers.L2` is 0.01 but that seemed to make the convergence much slower. Choosing a much smaller value did the trick.
Learning rate (Adam)	$0.001(0.99)^{\text{epoch}}$	I had originally used the Adam optimizer with its default learning rate, but manually changing the maximum learning rate seemed to help with convergence with higher epochs.
ResNet blocks	12	ResNet-34 includes 17 ResNet blocks, but I reduced the number to try to reduce epoch time. It still meets the desired results after 100 epochs.
# Filters	32, 64, 128	Similar to ResNet-34, as you go deeper in the network you have a higher number of filters for convolutional layers. I chose smaller values so that it would train faster.
Epochs	100	I guess this could be "set" using early stopping, but using the fixed value of 100 epochs was able to get both models to approximately 90% accuracy, which was good enough.

Figure 3: Hyperparameter selection justification

of which were greatly outperformed by running on Colab with a GPU. For example, an epoch that ran in roughly three minutes on my desktop ran in roughly 30 seconds on Colab.

- I did not implement cross-validation on a holdout set for hyperparameter tuning. Thus all of the hyperparameters were manually set as I tried to improve the model. This was due to time and hardware constraints, namely:
  - When training locally, the training time was very slow (a few hours).
  - When running on Google Colab, there is a timeout period, which means that I have to be constantly checking on the notebook (or have a script periodically ping the page). This was somewhat unreliable and required a lot of manual attention for long-running training sessions.
  - Tuning with `kerastuner.Hyperband` (as I did for the previous project) would require many more times the training time than a single train. Because of the short time span of this assignment and the little time that I had to work on it due to other classes, I was more focused on making larger improvements to the network (in order to meet the assignment goal on the test dataset) rather than making fine adjustments that would take a very long time to figure out by validation.

3 Results

Both the CIFAR-10 and CIFAR-100 were trained over 100 epochs. The classification accuracy on CIFAR-10 was 93.42%, and the top-5 classification accuracy on CIFAR-100 was 88.40%. This achieves the goal of 80% top-5 classification accuracy on CIFAR-100. According to benchmarks.ai [5], the top state-of-the-art models train at 99% test accuracy, which is far higher than what is achieved here. This might have been truly state-of-the-art around 2013 through 2017, in which the top accuracy was below 98%, but more recent models have achieved between 98% to 99% test accuracy.

This accuracy is comparable to that reported in [2] on CIFAR-10, which reported a 6.43% error (93.57% accuracy) with a 110-layer ResNet. The improved ResNet units (that my model is more closely based on) achieved a 5.46% error (94.54% accuracy) with a 164-layer ResNet architecture.

The time it takes to train the model on Google Colab with GPU enabled is roughly 43 seconds per epoch, so each model takes roughly 4300 seconds, or 71 minutes, to train.

4 Acknowledgments

- Yuval Ofek – Showed me the power of running on Colab GPUs rather than running it locally.
- Mark Kozykowski – Shared insights on some model improvements, e.g., image preprocessing with `keras.preprocessing.image.ImageDataGenerator` and using `tf.keras.callbacks.LearningRateScheduler` to adjust the maximum learning rate.

5 Source code

5.1 Setup

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import pickle

train_imgs = np.zeros((0, 3072))
train_lbls = []
```

5.2 Data entry (CIFAR-10)

```
# train datasets; split up into 6 training batches
for i in range(1, 6):
    with open('./data_batch_' + str(i), 'rb') as file:
        file_data = pickle.load(file, encoding='bytes')
        train_imgs = np.vstack((train_imgs, file_data[b'data']))
        train_lbls += file_data[b'labels']

train_lbls = tf.keras.utils.to_categorical(np.array(train_lbls))
train_imgs = train_imgs.reshape(-1, 3, 32, 32)
train_imgs = np.moveaxis(train_imgs, 1, -1)

# standardize data
train_imgs = (train_imgs - np.mean(train_imgs)) / np.std(train_imgs)

# test dataset
```

```
with open('./test_batch', 'rb') as file:
    file_data = pickle.load(file, encoding='bytes')
    test_imgs = file_data[b'data'].reshape(-1, 3, 32, 32)
    test_lbls = tf.keras.utils.to_categorical(np.array(file_data[b'labels']))
    test_imgs = np.moveaxis(test_imgs, 1, -1)
    test_imgs = (test_imgs - np.mean(test_imgs)) / np.std(test_imgs)
```

5.3 Data entry (CIFAR-100)

```
# train datasets
with open('./train', 'rb') as file:
    file_data = pickle.load(file, encoding='bytes')
    train_imgs = np.vstack((train_imgs, file_data[b'data']))
    train_lbls += file_data[b'fine_labels']

train_lbls = tf.keras.utils.to_categorical(np.array(train_lbls))
train_imgs = train_imgs.reshape(-1, 3, 32, 32)
train_imgs = np.moveaxis(train_imgs, 1, -1)

# standardize data
train_imgs = (train_imgs - np.mean(train_imgs)) / np.std(train_imgs)

# test dataset
with open('./test', 'rb') as file:
    file_data = pickle.load(file, encoding='bytes')
    test_imgs = file_data[b'data'].reshape(-1, 3, 32, 32)
    test_lbls = tf.keras.utils.to_categorical(np.array(file_data[b'fine_labels']))
    test_imgs = np.moveaxis(test_imgs, 1, -1)
    test_imgs = (test_imgs - np.mean(test_imgs)) / np.std(test_imgs)
```

5.4 Model

This is the code for the CIFAR-100 model. Two changes are made for the CIFAR-10 case:

- The last dense layer should have a width of 10.
- The model metric should be changed from top-5 accuracy to accuracy.

```
# initial number of filters for "first stage"; will be doubled twice
# as you progress deeper into the network
num_filters = 32
# for now, layers should be a multiple of 3
layers = 12

# input: 32x32x3 (3 = # color channels)
input = tf.keras.Input(shape=(32, 32, 3))

# do an initial convolution layer to increase dimensionality
x = tf.keras.layers.Conv2D(filters=num_filters,
                           kernel_size=7,
                           strides=1,
                           padding='same',
```

```
                            kernel_regularizer=tf.keras.regularizers.l2(1e-6),
                            kernel_initializer='he_normal')(input)

for i in range(layers):

    # increase number of filters twice as you go deeper in the network
    # 1x1 convolutional layer to change dimensionality
    if i > 0 and i % (layers / 3) == 0:
        num_filters *= 2
        x = tf.keras.layers.Conv2D(filters=num_filters,
                                   kernel_size=1,
                                   padding='same',
                                   kernel_regularizer=tf.keras.regularizers.l2(1e-6),
                                   kernel_initializer='he_normal')(x)

    # first batchnorm, activation, conv2d
    unit = tf.keras.layers.BatchNormalization()(x)
    unit = tf.keras.layers.ReLU()(unit)

    # in first layer of a "block," no skip connection and use 2x2 strides to
    # decrease image dimensions, see ResNet-34 diagram; for other units, add a
    # skip connection
    if i > 0 and i % (layers / 3) == 0:
        unit = tf.keras.layers.Conv2D(filters=num_filters,
                                      kernel_size=3,
                                      padding='same',
                                      strides=2,
                                      kernel_regularizer=tf.keras.regularizers.l2(1e-6)
                                          ,
                                      kernel_initializer='he_normal')(unit)
        x = unit
    else:
        unit = tf.keras.layers.Conv2D(filters=num_filters,
                                      kernel_size=3,
                                      padding='same',
                                      kernel_regularizer=tf.keras.regularizers.l2(1e-6)
                                          ,
                                      kernel_initializer='he_normal')(unit)
        x = tf.keras.layers.Add()([x, unit])

    # second batchnorm, activation, conv2d
    unit = tf.keras.layers.BatchNormalization()(x)
    unit = tf.keras.layers.ReLU()(unit)
    unit = tf.keras.layers.Conv2D(filters=num_filters,
                                  kernel_size=3,
                                  padding='same',
                                  kernel_initializer='he_normal',
                                  kernel_regularizer=tf.keras.regularizers.l2(1e-6))(
                                      unit)
    unit = tf.keras.layers.Dropout(rate=0.1)(unit)
    x = tf.keras.layers.Add()([x, unit])

# final part: batchnorm, pooling, dense layer (logits for softmax)
x = tf.keras.layers.BatchNormalization()(x)
x = tf.keras.layers.AveragePooling2D(pool_size=8)(x)
x = tf.keras.layers.Flatten()(x)

# for CIFAR-100, units=100; for CIFAR-10, units=10
```

```
x = tf.keras.layers.Dense(units=100,
                          kernel_initializer='he_normal',
                          kernel_regularizer=tf.keras.regularizers.L1L2(l2=1e-6))(x)

model = tf.keras.models.Model(inputs=[input], outputs=x)

# set up model loss and optimizer
# for CIFAR-100, tf.keras.metrics.TopKCategoricalAccuracy(k=5);
# for CIFAR-10, use 'accuracy'
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
    loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
    metrics=[tf.keras.metrics.TopKCategoricalAccuracy(k=5)])
```

5.5 Image preprocessing/augmentation and training

```
# feature preprocessing
datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    rotation_range=30,
    width_shift_range=0.1,
    height_shift_range=0.1,
    fill_mode='nearest',
    horizontal_flip=True,
    vertical_flip=True,
)

# training
datagen.fit(train_imgs)

def learning_rate_scheduler(epoch):
    return 1e-3 * 0.99**epoch

model.fit_generator(datagen.flow(train_imgs, train_lbls),
                    callbacks=[tf.keras.callbacks.LearningRateScheduler(
                        learning_rate_scheduler)],
                    epochs=100, verbose=1)
```

5.6 Model evaluation

```
print('Evaluating on test dataset')
model.evaluate(test_imgs, test_lbls)
```

6 Code output

6.1 CIFAR-10

```
Epoch 1/100
1563/1563 [==============================] - 44s 28ms/step - loss: 1.7498 - accuracy: 0.3513
Epoch 2/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.4927 - accuracy: 0.4610
Epoch 3/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.3685 - accuracy: 0.5118
Epoch 4/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.2806 - accuracy: 0.5473
Epoch 5/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.2195 - accuracy: 0.5708
Epoch 6/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.1637 - accuracy: 0.5912
Epoch 7/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.1174 - accuracy: 0.6063
Epoch 8/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.0714 - accuracy: 0.6274
Epoch 9/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.0310 - accuracy: 0.6421
Epoch 10/100
1563/1563 [==============================] - 43s 28ms/step - loss: 0.9921 - accuracy: 0.6576
Epoch 11/100
1563/1563 [==============================] - 43s 28ms/step - loss: 0.9654 - accuracy: 0.6647
Epoch 12/100
1563/1563 [==============================] - 43s 28ms/step - loss: 0.9349 - accuracy: 0.6783
Epoch 13/100
1563/1563 [==============================] - 43s 28ms/step - loss: 0.9045 - accuracy: 0.6888
Epoch 14/100
1563/1563 [==============================] - 43s 28ms/step - loss: 0.8809 - accuracy: 0.6986
Epoch 15/100
1563/1563 [==============================] - 43s 28ms/step - loss: 0.8495 - accuracy: 0.7096
Epoch 16/100
1563/1563 [==============================] - 44s 28ms/step - loss: 0.8235 - accuracy: 0.7209
Epoch 17/100
1563/1563 [==============================] - 43s 28ms/step - loss: 0.8076 - accuracy: 0.7244
Epoch 18/100
1563/1563 [==============================] - 44s 28ms/step - loss: 0.7935 - accuracy: 0.7325
Epoch 19/100
1563/1563 [==============================] - 44s 28ms/step - loss: 0.7762 - accuracy: 0.7381
Epoch 20/100
1563/1563 [==============================] - 44s 28ms/step - loss: 0.7591 - accuracy: 0.7458
Epoch 21/100
1563/1563 [==============================] - 44s 28ms/step - loss: 0.7415 - accuracy: 0.7502
Epoch 22/100
1563/1563 [==============================] - 44s 28ms/step - loss: 0.7253 - accuracy: 0.7580
Epoch 23/100
1563/1563 [==============================] - 44s 28ms/step - loss: 0.7176 - accuracy: 0.7641
Epoch 24/100
1563/1563 [==============================] - 44s 28ms/step - loss: 0.7008 - accuracy: 0.7674
Epoch 25/100
1563/1563 [==============================] - 44s 28ms/step - loss: 0.6915 - accuracy: 0.7706
Epoch 26/100
1563/1563 [==============================] - 44s 28ms/step - loss: 0.6771 - accuracy: 0.7773
Epoch 27/100
1563/1563 [==============================] - 44s 28ms/step - loss: 0.6628 - accuracy: 0.7834
Epoch 28/100
1563/1563 [==============================] - 44s 28ms/step - loss: 0.6594 - accuracy: 0.7832
Epoch 29/100
1563/1563 [==============================] - 43s 28ms/step - loss: 0.6415 - accuracy: 0.7920
Epoch 30/100
1563/1563 [==============================] - 44s 28ms/step - loss: 0.6367 - accuracy: 0.7928
Epoch 31/100
1563/1563 [==============================] - 44s 28ms/step - loss: 0.6256 - accuracy: 0.7966
Epoch 32/100
1563/1563 [==============================] - 44s 28ms/step - loss: 0.6177 - accuracy: 0.7991
Epoch 33/100
1563/1563 [==============================] - 43s 28ms/step - loss: 0.6069 - accuracy: 0.8015
Epoch 34/100
1563/1563 [==============================] - 43s 28ms/step - loss: 0.6019 - accuracy: 0.8047
Epoch 35/100
1563/1563 [==============================] - 43s 28ms/step - loss: 0.5895 - accuracy: 0.8105
Epoch 36/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.5819 - accuracy: 0.8123
Epoch 37/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.5758 - accuracy: 0.8130
Epoch 38/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.5690 - accuracy: 0.8167
Epoch 39/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.5584 - accuracy: 0.8218
Epoch 40/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.5511 - accuracy: 0.8243
Epoch 41/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.5488 - accuracy: 0.8244
Epoch 42/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.5448 - accuracy: 0.8250
Epoch 43/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.5385 - accuracy: 0.8288
Epoch 44/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.5316 - accuracy: 0.8294
Epoch 45/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.5247 - accuracy: 0.8320
Epoch 46/100
```

```
1563/1563 [==============================] - 43s 27ms/step - loss: 0.5177 - accuracy: 0.8363
Epoch 47/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.5098 - accuracy: 0.8400
Epoch 48/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.5073 - accuracy: 0.8389
Epoch 49/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.5002 - accuracy: 0.8432
Epoch 50/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.4958 - accuracy: 0.8437
Epoch 51/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.4911 - accuracy: 0.8472
Epoch 52/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.4885 - accuracy: 0.8469
Epoch 53/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.4814 - accuracy: 0.8491
Epoch 54/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.4754 - accuracy: 0.8515
Epoch 55/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.4726 - accuracy: 0.8533
Epoch 56/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.4667 - accuracy: 0.8543
Epoch 57/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.4622 - accuracy: 0.8585
Epoch 58/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.4579 - accuracy: 0.8576
Epoch 59/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.4582 - accuracy: 0.8573
Epoch 60/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.4521 - accuracy: 0.8603
Epoch 61/100
1563/1563 [==============================] - 42s 27ms/step - loss: 0.4472 - accuracy: 0.8610
Epoch 62/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.4441 - accuracy: 0.8631
Epoch 63/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.4410 - accuracy: 0.8638
Epoch 64/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.4389 - accuracy: 0.8651
Epoch 65/100
1563/1563 [==============================] - 42s 27ms/step - loss: 0.4364 - accuracy: 0.8673
Epoch 66/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.4315 - accuracy: 0.8671
Epoch 67/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.4293 - accuracy: 0.8677
Epoch 68/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.4242 - accuracy: 0.8700
Epoch 69/100
1563/1563 [==============================] - 42s 27ms/step - loss: 0.4187 - accuracy: 0.8733
Epoch 70/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.4166 - accuracy: 0.8725
Epoch 71/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.4103 - accuracy: 0.8754
Epoch 72/100
1563/1563 [==============================] - 42s 27ms/step - loss: 0.4078 - accuracy: 0.8766
Epoch 73/100
1563/1563 [==============================] - 42s 27ms/step - loss: 0.4074 - accuracy: 0.8768
Epoch 74/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.4024 - accuracy: 0.8783
Epoch 75/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.3995 - accuracy: 0.8801
Epoch 76/100
1563/1563 [==============================] - 42s 27ms/step - loss: 0.3937 - accuracy: 0.8827
Epoch 77/100
1563/1563 [==============================] - 42s 27ms/step - loss: 0.3942 - accuracy: 0.8805
Epoch 78/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.3892 - accuracy: 0.8846
Epoch 79/100
1563/1563 [==============================] - 42s 27ms/step - loss: 0.3878 - accuracy: 0.8825
Epoch 80/100
1563/1563 [==============================] - 42s 27ms/step - loss: 0.3851 - accuracy: 0.8838
Epoch 81/100
1563/1563 [==============================] - 42s 27ms/step - loss: 0.3828 - accuracy: 0.8841
Epoch 82/100
1563/1563 [==============================] - 42s 27ms/step - loss: 0.3798 - accuracy: 0.8863
Epoch 83/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.3781 - accuracy: 0.8871
Epoch 84/100
1563/1563 [==============================] - 42s 27ms/step - loss: 0.3761 - accuracy: 0.8868
Epoch 85/100
1563/1563 [==============================] - 42s 27ms/step - loss: 0.3669 - accuracy: 0.8894
Epoch 86/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.3693 - accuracy: 0.8895
Epoch 87/100
1563/1563 [==============================] - 42s 27ms/step - loss: 0.3637 - accuracy: 0.8911
Epoch 88/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.3634 - accuracy: 0.8912
Epoch 89/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.3607 - accuracy: 0.8918
Epoch 90/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.3568 - accuracy: 0.8942
Epoch 91/100
1563/1563 [==============================] - 42s 27ms/step - loss: 0.3510 - accuracy: 0.8974
Epoch 92/100
```

```
1563/1563 [==============================] - 43s 27ms/step - loss: 0.3525 - accuracy: 0.8946
Epoch 93/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.3511 - accuracy: 0.8954
Epoch 94/100
1563/1563 [==============================] - 42s 27ms/step - loss: 0.3482 - accuracy: 0.8975
Epoch 95/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.3459 - accuracy: 0.8988
Epoch 96/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.3457 - accuracy: 0.8989
Epoch 97/100
1563/1563 [==============================] - 42s 27ms/step - loss: 0.3423 - accuracy: 0.8996
Epoch 98/100
1563/1563 [==============================] - 42s 27ms/step - loss: 0.3361 - accuracy: 0.9016
Epoch 99/100
1563/1563 [==============================] - 42s 27ms/step - loss: 0.3396 - accuracy: 0.9005
Epoch 100/100
1563/1563 [==============================] - 43s 27ms/step - loss: 0.3362 - accuracy: 0.9022
Evaluating on test dataset
313/313 [==============================] - 3s 9ms/step - loss: 0.2387 - accuracy: 0.9342
[0.23870298266410828, 0.9341999888420105]
```

6.2 CIFAR-100

```
Epoch 1/100
1563/1563 [==============================] - 42s 27ms/step - loss: 4.0681 - top_k_categorical_accuracy: 0.2525
Epoch 2/100
1563/1563 [==============================] - 42s 27ms/step - loss: 3.6597 - top_k_categorical_accuracy: 0.3778
Epoch 3/100
1563/1563 [==============================] - 44s 28ms/step - loss: 3.3683 - top_k_categorical_accuracy: 0.4632
Epoch 4/100
1563/1563 [==============================] - 44s 28ms/step - loss: 3.1648 - top_k_categorical_accuracy: 0.5156
Epoch 5/100
1563/1563 [==============================] - 44s 28ms/step - loss: 2.9978 - top_k_categorical_accuracy: 0.5584
Epoch 6/100
1563/1563 [==============================] - 44s 28ms/step - loss: 2.8675 - top_k_categorical_accuracy: 0.5908
Epoch 7/100
1563/1563 [==============================] - 43s 28ms/step - loss: 2.7648 - top_k_categorical_accuracy: 0.6177
Epoch 8/100
1563/1563 [==============================] - 43s 27ms/step - loss: 2.6732 - top_k_categorical_accuracy: 0.6388
Epoch 9/100
1563/1563 [==============================] - 43s 28ms/step - loss: 2.5991 - top_k_categorical_accuracy: 0.6559
Epoch 10/100
1563/1563 [==============================] - 43s 28ms/step - loss: 2.5330 - top_k_categorical_accuracy: 0.6707
Epoch 11/100
1563/1563 [==============================] - 43s 28ms/step - loss: 2.4589 - top_k_categorical_accuracy: 0.6873
Epoch 12/100
1563/1563 [==============================] - 43s 28ms/step - loss: 2.3976 - top_k_categorical_accuracy: 0.6997
Epoch 13/100
1563/1563 [==============================] - 43s 28ms/step - loss: 2.3379 - top_k_categorical_accuracy: 0.7119
Epoch 14/100
1563/1563 [==============================] - 43s 28ms/step - loss: 2.2853 - top_k_categorical_accuracy: 0.7228
Epoch 15/100
1563/1563 [==============================] - 43s 27ms/step - loss: 2.2403 - top_k_categorical_accuracy: 0.7344
Epoch 16/100
1563/1563 [==============================] - 43s 28ms/step - loss: 2.1943 - top_k_categorical_accuracy: 0.7417
Epoch 17/100
1563/1563 [==============================] - 43s 28ms/step - loss: 2.1394 - top_k_categorical_accuracy: 0.7536
Epoch 18/100
1563/1563 [==============================] - 43s 28ms/step - loss: 2.0985 - top_k_categorical_accuracy: 0.7650
Epoch 19/100
1563/1563 [==============================] - 43s 28ms/step - loss: 2.0604 - top_k_categorical_accuracy: 0.7693
Epoch 20/100
1563/1563 [==============================] - 43s 28ms/step - loss: 2.0263 - top_k_categorical_accuracy: 0.7773
Epoch 21/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.9891 - top_k_categorical_accuracy: 0.7846
Epoch 22/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.9609 - top_k_categorical_accuracy: 0.7893
Epoch 23/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.9264 - top_k_categorical_accuracy: 0.7963
Epoch 24/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.8968 - top_k_categorical_accuracy: 0.8026
Epoch 25/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.8769 - top_k_categorical_accuracy: 0.8055
Epoch 26/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.8459 - top_k_categorical_accuracy: 0.8112
Epoch 27/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.8216 - top_k_categorical_accuracy: 0.8134
Epoch 28/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.7960 - top_k_categorical_accuracy: 0.8205
Epoch 29/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.7752 - top_k_categorical_accuracy: 0.8224
Epoch 30/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.7529 - top_k_categorical_accuracy: 0.8265
Epoch 31/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.7329 - top_k_categorical_accuracy: 0.8301
Epoch 32/100
```

```
1563/1563 [==============================] - 43s 27ms/step - loss: 1.7136 - top_k_categorical_accuracy: 0.8319
Epoch 33/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.6925 - top_k_categorical_accuracy: 0.8377
Epoch 34/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.6768 - top_k_categorical_accuracy: 0.8409
Epoch 35/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.6568 - top_k_categorical_accuracy: 0.8439
Epoch 36/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.6362 - top_k_categorical_accuracy: 0.8465
Epoch 37/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.6228 - top_k_categorical_accuracy: 0.8509
Epoch 38/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.6114 - top_k_categorical_accuracy: 0.8527
Epoch 39/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.5943 - top_k_categorical_accuracy: 0.8555
Epoch 40/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.5731 - top_k_categorical_accuracy: 0.8586
Epoch 41/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.5627 - top_k_categorical_accuracy: 0.8592
Epoch 42/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.5467 - top_k_categorical_accuracy: 0.8631
Epoch 43/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.5284 - top_k_categorical_accuracy: 0.8651
Epoch 44/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.5174 - top_k_categorical_accuracy: 0.8640
Epoch 45/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.4951 - top_k_categorical_accuracy: 0.8711
Epoch 46/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.4918 - top_k_categorical_accuracy: 0.8701
Epoch 47/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.4754 - top_k_categorical_accuracy: 0.8723
Epoch 48/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.4596 - top_k_categorical_accuracy: 0.8756
Epoch 49/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.4506 - top_k_categorical_accuracy: 0.8773
Epoch 50/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.4426 - top_k_categorical_accuracy: 0.8788
Epoch 51/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.4234 - top_k_categorical_accuracy: 0.8806
Epoch 52/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.4232 - top_k_categorical_accuracy: 0.8805
Epoch 53/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.4075 - top_k_categorical_accuracy: 0.8847
Epoch 54/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.4013 - top_k_categorical_accuracy: 0.8854
Epoch 55/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.3845 - top_k_categorical_accuracy: 0.8884
Epoch 56/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.3836 - top_k_categorical_accuracy: 0.8878
Epoch 57/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.3690 - top_k_categorical_accuracy: 0.8905
Epoch 58/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.3541 - top_k_categorical_accuracy: 0.8906
Epoch 59/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.3458 - top_k_categorical_accuracy: 0.8937
Epoch 60/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.3425 - top_k_categorical_accuracy: 0.8939
Epoch 61/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.3249 - top_k_categorical_accuracy: 0.8953
Epoch 62/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.3209 - top_k_categorical_accuracy: 0.8963
Epoch 63/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.3073 - top_k_categorical_accuracy: 0.8980
Epoch 64/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.2956 - top_k_categorical_accuracy: 0.9012
Epoch 65/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.2907 - top_k_categorical_accuracy: 0.9019
Epoch 66/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.2829 - top_k_categorical_accuracy: 0.9024
Epoch 67/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.2730 - top_k_categorical_accuracy: 0.9028
Epoch 68/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.2616 - top_k_categorical_accuracy: 0.9049
Epoch 69/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.2598 - top_k_categorical_accuracy: 0.9054
Epoch 70/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.2408 - top_k_categorical_accuracy: 0.9087
Epoch 71/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.2406 - top_k_categorical_accuracy: 0.9088
Epoch 72/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.2295 - top_k_categorical_accuracy: 0.9095
Epoch 73/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.2252 - top_k_categorical_accuracy: 0.9109
Epoch 74/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.2168 - top_k_categorical_accuracy: 0.9125
Epoch 75/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.2135 - top_k_categorical_accuracy: 0.9114
Epoch 76/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.1976 - top_k_categorical_accuracy: 0.9138
Epoch 77/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.1951 - top_k_categorical_accuracy: 0.9131
Epoch 78/100
```

```
1563/1563 [==============================] - 43s 28ms/step - loss: 1.1925 - top_k_categorical_accuracy: 0.9139
Epoch 79/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.1755 - top_k_categorical_accuracy: 0.9180
Epoch 80/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.1669 - top_k_categorical_accuracy: 0.9181
Epoch 81/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.1592 - top_k_categorical_accuracy: 0.9186
Epoch 82/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.1631 - top_k_categorical_accuracy: 0.9184
Epoch 83/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.1433 - top_k_categorical_accuracy: 0.9211
Epoch 84/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.1510 - top_k_categorical_accuracy: 0.9204
Epoch 85/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.1361 - top_k_categorical_accuracy: 0.9218
Epoch 86/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.1320 - top_k_categorical_accuracy: 0.9218
Epoch 87/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.1281 - top_k_categorical_accuracy: 0.9226
Epoch 88/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.1182 - top_k_categorical_accuracy: 0.9245
Epoch 89/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.1142 - top_k_categorical_accuracy: 0.9263
Epoch 90/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.1017 - top_k_categorical_accuracy: 0.9282
Epoch 91/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.1026 - top_k_categorical_accuracy: 0.9276
Epoch 92/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.0994 - top_k_categorical_accuracy: 0.9276
Epoch 93/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.0958 - top_k_categorical_accuracy: 0.9259
Epoch 94/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.0832 - top_k_categorical_accuracy: 0.9289
Epoch 95/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.0741 - top_k_categorical_accuracy: 0.9303
Epoch 96/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.0705 - top_k_categorical_accuracy: 0.9303
Epoch 97/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.0670 - top_k_categorical_accuracy: 0.9307
Epoch 98/100
1563/1563 [==============================] - 43s 28ms/step - loss: 1.0651 - top_k_categorical_accuracy: 0.9313
Epoch 99/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.0631 - top_k_categorical_accuracy: 0.9323
Epoch 100/100
1563/1563 [==============================] - 43s 27ms/step - loss: 1.0502 - top_k_categorical_accuracy: 0.9328
Evaluating on test dataset
313/313 [==============================] - 3s 8ms/step - loss: 1.5470 - top_k_categorical_accuracy: 0.8840
[1.5469876527786255, 0.8840000033378601]
```

References

[1] https://www.cs.toronto.edu/~kriz/cifar.html

[2] He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

[3] He, Kaiming, et al. "Identity mappings in deep residual networks." *European conference on computer vision*. Springer, Cham, 2016.

[4] Recombination of Artificial Neural Networks - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/ResNet-neural-network-architecture-ResNet-34-pictured-image-from-11_fig6_330400293 [accessed 1 Oct, 2020]

[5] https://benchmarks.ai/cifar-10