Amaan Rahman

ECE 472: Deep Learning

Professor Curro

Assignment 4: CIFAR Image Classification

CIFAR - 10:

Various experiments were performed with the CIFAR-10 image classification task. Initially a VGG-19 architecture was implemented with some modifications in the depth of the model to be suitable for CIFAR-10. What was interesting was how the loss spiked and the accuracy plummeted.

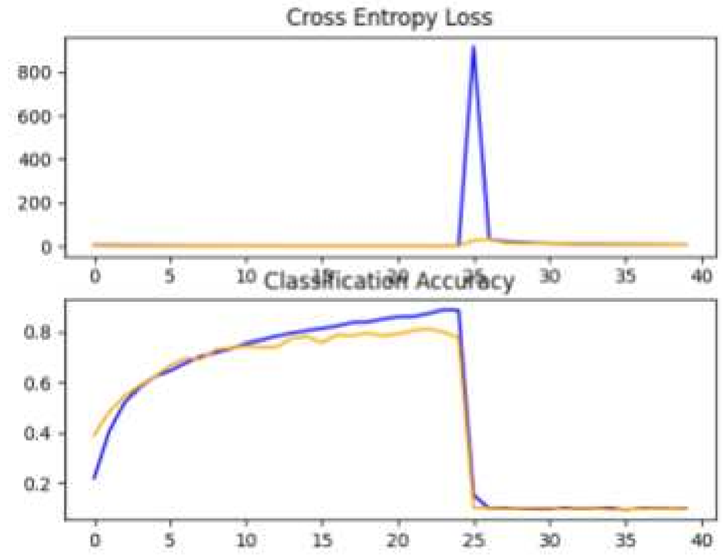


Figure 1: VGG-19-Like Model presenting vanishing gradient problem

Blue: train, **Yellow:** validation

One way I mitigated this issue was through introducing a **learning rate scheduler** to shave off the learning rate after a fixed number of epochs so the vanishing gradient problem could be mitigated.

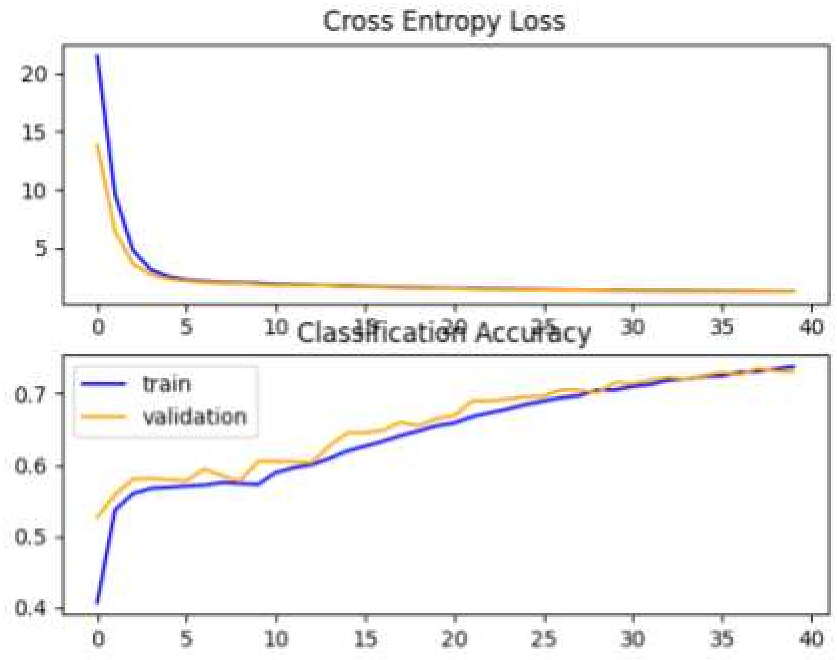


Figure 2: VGG-19-Like model with Learning Rate Scheduler

After a while tampering with the VGG like model, my test accuracy kept converging to as high as 70%. I decided to move on to a **ResNet** architecture [1], [2].

Considerable improvements were observable: fewer parameters, faster training time, and larger test accuracy. I also introduced preprocessing techniques such as random cropping and flipping the input image to model. At first my initial implementation was a **ResNet-50** model and that yielded 75%, however it was unoptimized and poorly implemented. I then tested out a **ResNet-20** model, which yielded test accuracy of about 80%. A nuisance that I noticed was that the validation accuracy would converge faster than the training accuracy (same with the loss). Of course this is due to the model overfitting with the training data. I implemented a strong dropout procedure with my **ResNet-18** model; I applied dropouts with rates of 0.5 for the residual blocks and 0.3 for the fully connected classification layer. I also implemented the built-in tensorflow function: ReducedLROnPlateau to shave off the learning rate by 1/10th after observing no noticeable change in the validation loss over 10 trials. This yielded my best results thus far of a test accuracy **86.75%**.

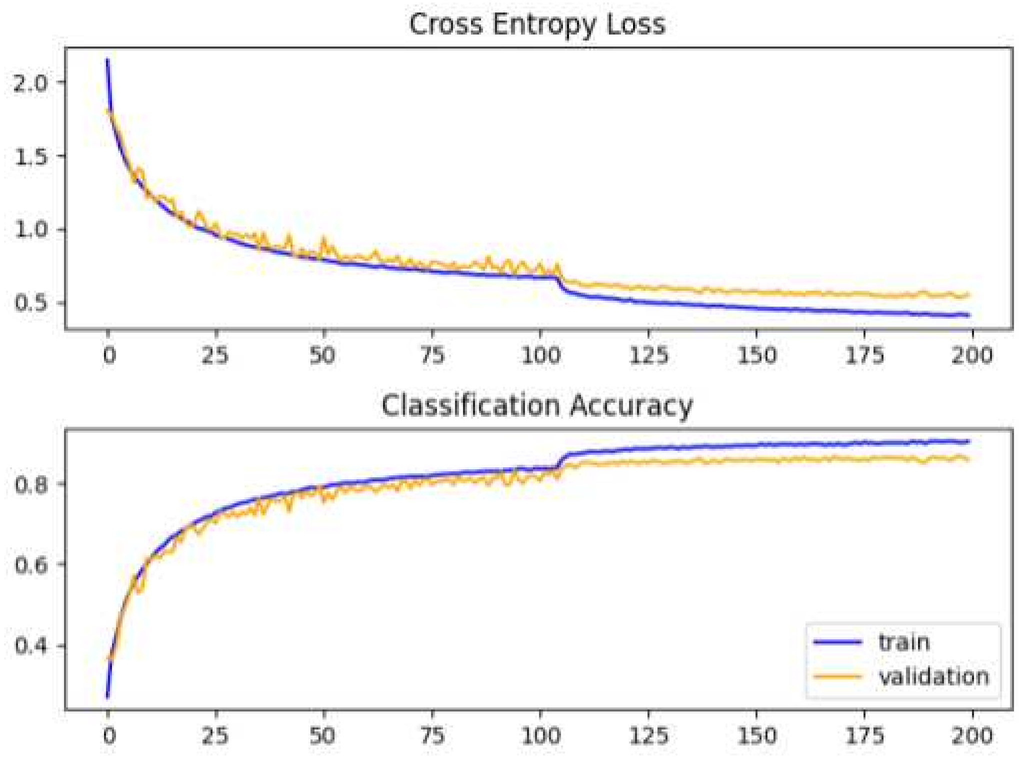
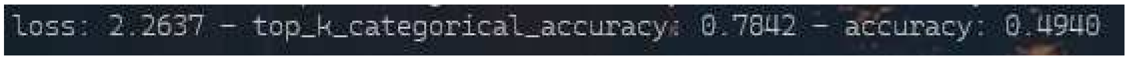


Figure 3: ResNet-18 with strong dropout; learning rate reduction effect can be seen around 105th epoch

Other miscellaneous experiments I conducted were experimenting with a cyclical learning rate scheduler to replicate the super convergence paper [3], [4]; this didn't work out so well because I wasn't optimizing it well with this specific use case. Another experiment I conducted was using a sparse gate block within a residual block [5]. This was an interesting experiment because the introduction of this block would discard residuals, analogous to a dropout function but for an entire residual block. I couldn't properly take advantage of this technique because I was experimenting on not-so-deep residual networks. This also motivated me to take a look at highway networks, however I didn't get the chance to implement anything of that sort.

CIFAR-100:

I implemented various **ResNet** architectures such as **ResNet-110**, **ResNet-50**, and **ResNet-18**. The former 2 models yielded horrible convergence (35% or 50% convergence for validation set after 30 or 60 epochs out of 200 epochs), however **ResNet-18** fared well. I introduced **sparse dropout** after the shortcut addition within the identity residual block preventing overfitting and increasing generalization. Unfortunately, for this part of the project my accuracy graph is not based on the "top-5-accuracy" metric so there is no graph for it :(. The top-5-accuracy for CIFAR-100 with **ResNet-18** was **78.42%.**



```
loss: 2.2637 - top_k_categorical_accuracy: 0.7842 - accuracy: 0.4940
```

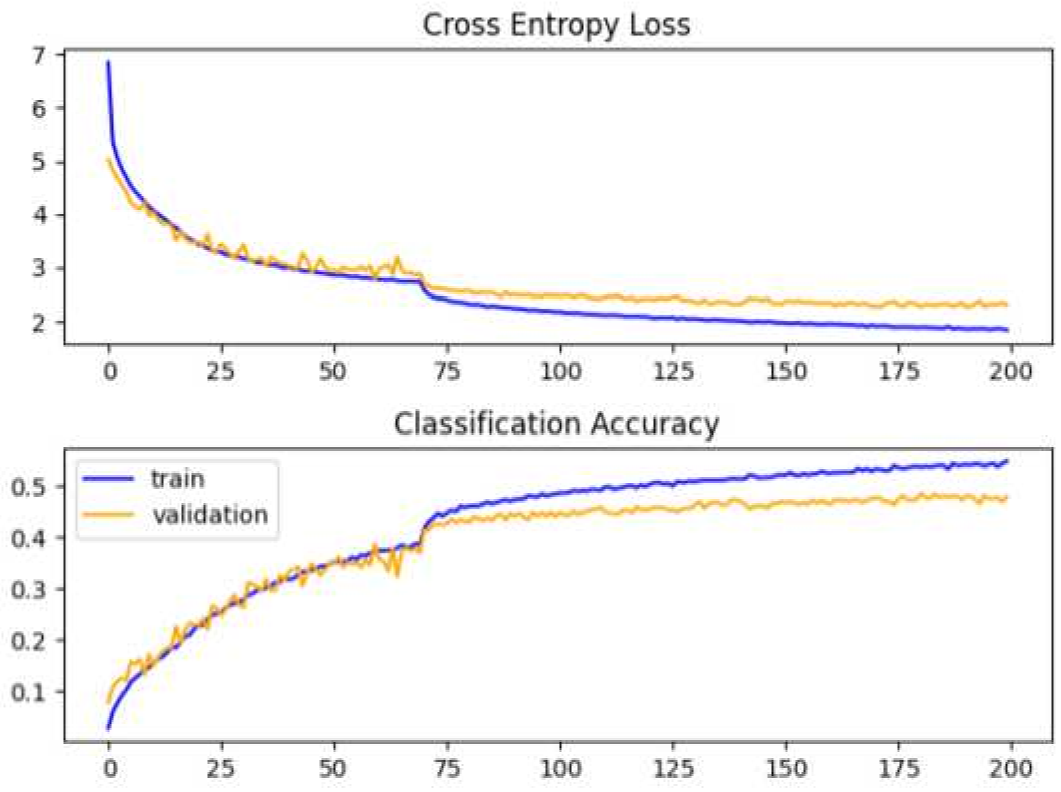


Figure 4: ResNet-18 model against **CIFAR-100** using strong sparse dropout rate of 0.5 and ReduceLROnPlateau

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 770–778. doi: 10.1109/CVPR.2016.90.

[2] "How to code your ResNet from scratch in Tensorflow?," *Analytics Vidhya*, Aug. 26, 2021. https://www.analyticsvidhya.com/blog/2021/08/how-to-code-your-resnet-from-scratch-in-tensorflow/ (accessed Oct. 11, 2021).

[3] L. N. Smith, "Cyclical Learning Rates for Training Neural Networks," *arXiv:1506.01186 [cs]*, Apr. 2017, Accessed: Oct. 11, 2021. [Online]. Available: http://arxiv.org/abs/1506.01186

[4] C. T. Bs. H. MIAP, "Super Convergence with Cyclical Learning Rates in TensorFlow," *Medium*, Jan. 31, 2021. https://towardsdatascience.com/super-convergence-with-cyclical-learning-rates-in-tensorflow-c1932b858252 (accessed Oct. 11, 2021).

[5] X. Yu, Z. Yu, and S. Ramalingam, "Learning Strict Identity Mappings in Deep Residual Networks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, Jun. 2018, pp. 4432–4440. doi: 10.1109/CVPR.2018.00466.

```python
from tensorflow.python.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from matplotlib import pyplot as plt
from resnet import ResNet_N
from darse import Parser
from os import getpid

BATCH_SIZE = 128
EPOCHS = 200

# CIFAR_10
dataset_10 = {
    "train-01": "/zooper2/amaan.rahman/ECE472-DeepLearning/datasets/CIFAR10_DATASET/pkl/data_batch_
1",
    "train-02": "/zooper2/amaan.rahman/ECE472-DeepLearning/datasets/CIFAR10_DATASET/pkl/data_batch_
2",
    "train-03": "/zooper2/amaan.rahman/ECE472-DeepLearning/datasets/CIFAR10_DATASET/pkl/data_batch_
3",
    "train-04": "/zooper2/amaan.rahman/ECE472-DeepLearning/datasets/CIFAR10_DATASET/pkl/data_batch_
4",
    "train-05": "/zooper2/amaan.rahman/ECE472-DeepLearning/datasets/CIFAR10_DATASET/pkl/data_batch_
5",
    "test": "/zooper2/amaan.rahman/ECE472-DeepLearning/datasets/CIFAR10_DATASET/pkl/test_batch",
}

# CIFAR_100
dataset_100 = {
    "train": "/zooper2/amaan.rahman/ECE472-DeepLearning/datasets/CIFAR100_DATASET/pkl/train",
    "test": "/zooper2/amaan.rahman/ECE472-DeepLearning/datasets/CIFAR100_DATASET/pkl/test",
}

# https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar
-10-photo-classification/
def plot_diagnostics(history):
    # plot loss
    plt.subplot(211)
    plt.title("Cross Entropy Loss")
    plt.plot(history.history["loss"], color="blue", label="train")
    plt.plot(history.history["val_loss"], color="orange", label="validation")
    # plot accuracy
    plt.subplot(212)
    plt.title("Classification Accuracy")
    plt.plot(history.history["accuracy"], color="blue", label="train")
    plt.plot(history.history["val_accuracy"], color="orange", label="validation")
    plt.legend()
    plt.tight_layout()
    # save plot to file
    plt.savefig(
        "/zooper2/amaan.rahman/ECE472-DeepLearning/assign4/E1_cifar100_plot_"
        + str(getpid())
        + ".png"
    )
    plt.close()


def gen_data(cifar_type):
    dataset = dataset_100 if cifar_type == "CIFAR_100" else dataset_10

    # parse
    cifar_parser = Parser(dataset, cifar_type)
    train, test = cifar_parser.parse()
```

```python
    train_data, train_labels = train
    test_data, test_labels = test

    # convert labels to one-hot format
    train_labels = to_categorical(train_labels)
    test_labels = to_categorical(test_labels)

    return train_data, train_labels, test_data, test_labels


def main():
    # dataset parse
    train_data, train_labels, test_data, test_labels = gen_data("CIFAR_100")
    STEPS = 0.8 * train_data.shape[0] // BATCH_SIZE

    # model init
    model = ResNet_N(
        in_shape=(test_data.shape[1], test_data.shape[2], 3),
        layers=[2, 2, 2, 2],
        classes=100,
    )
    model.summary()

    # model compile
    # https://towardsdatascience.com/super-convergence-with-cyclical-learning-ra
tes-in-tensorflow-c1932b858252
    # https://arxiv.org/pdf/1506.01186.pdf
    model.compile(
        optimizer=Adam(),
        loss="categorical_crossentropy",
        metrics=["top_k_categorical_accuracy", "accuracy"],
    )

    # fit
    callback = ReduceLROnPlateau(monitor="val_loss", min_lr=1e-4, verbose=1)
    history = model.fit(
        x=train_data,
        y=train_labels,
        batch_size=BATCH_SIZE,
        epochs=EPOCHS,
        steps_per_epoch=STEPS,
        callbacks=[callback],
        validation_split=0.2,
    )

    # evaluate
    model.evaluate(x=test_data, y=test_labels)
    plot_diagnostics(history)


if __name__ == "__main__":
    main()
```

```
from tensorflow.keras.models import Model
from tensorflow.keras import regularizers
from tensorflow.keras.layers import (
    Dense,
    Conv2D,
    Dropout,
    Flatten,
    BatchNormalization,
    Activation,
    Add,
    Input,
    ZeroPadding2D,
    AveragePooling2D,
)
from tensorflow.python.keras.layers.core import SpatialDropout2D
from tensorflow.keras.layers.experimental.preprocessing import RandomCrop, Rando
mFlip
from tensorflow.python.keras.layers.preprocessing.image_preprocessing import HOR
IZONTAL
import numpy as np

BOTTLENECK = False  # enable this for CIFAR-100, sorry for the bad code design :
/

# https://www.analyticsvidhya.com/blog/2021/08/how-to-code-your-resnet-from-scra
tch-in-tensorflow/
# https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/He_Deep_Resi
dual_Learning_CVPR_2016_paper.pdf
def basic_blk(input, k, f, s):
    out = BatchNormalization(axis=3, momentum=0.9)(input)
    out = Activation("elu")(out)
    out = Conv2D(
        filters=f,
        kernel_size=k[0],
        kernel_initializer="he_normal",
        kernel_regularizer=regularizers.l2(l2=0.0001),
        padding="same",
        strides=s,
    )(out)

    out = BatchNormalization(axis=3, momentum=0.9)(out)
    out = Activation("elu")(out)
    out = Conv2D(
        filters=f,
        kernel_size=k[1],
        kernel_initializer="he_normal",
        kernel_regularizer=regularizers.l2(l2=0.0001),
        padding="same",
    )(out)

    if BOTTLENECK:
        out = BatchNormalization(axis=3, momentum=0.9)(out)
        out = Activation("elu")(out)
        out = Conv2D(
            filters=(4 * f),
            kernel_size=k[2],
            kernel_initializer="he_normal",
            kernel_regularizer=regularizers.l2(l2=0.00001),
            padding="same",
        )(out)
    return out
```

```

def ident_blk(input, filter_depth):
    ff_input = input
    out = Dropout(0.5)(input)
    out = basic_blk(out, (3, 3), filter_depth, (1, 1))
    out = Add()([out, ff_input])
    out = SpatialDropout2D(0.5)(out)
    return out


def conv_blk(input, filter_depth, stride):
    ff_input = input
    out = basic_blk(input, (3, 3), filter_depth, stride)
    ff_input = BatchNormalization(axis=3, momentum=0.9)(ff_input)
    ff_input = Activation("elu")(ff_input)
    ff_input = Conv2D(
        filter_depth,
        kernel_size=(1, 1),
        kernel_initializer="he_normal",
        kernel_regularizer=regularizers.l2(l2=0.00001),
        strides=stride,
        padding="same",
    )(ff_input)
    out = Add()([out, ff_input])
    return out


def res_blk(x, filter_depth, num_layers, init_stride):
    x = conv_blk(x, filter_depth, init_stride)
    for i in range(num_layers - 1):
        x = ident_blk(x, filter_depth)
    return x


def ResNet_N(in_shape, layers, classes):
    filter_depth = 64
    input = Input(in_shape)

    # Preprocessing method: RANDOM CROP
    x = ZeroPadding2D(padding=(4, 4))(input)
    x = RandomCrop(32, 32)(x)
    x = RandomFlip(mode=HORIZONTAL)(x)

    # model
    x = Conv2D(
        filter_depth,
        kernel_size=3,
        kernel_initializer="he_normal",
        kernel_regularizer=regularizers.l2(l2=0.00001),
        padding="same",
        strides=2,
    )(x)

    x = BatchNormalization(axis=3, momentum=0.9)(x)
    x = Activation("elu")(x)

    x = res_blk(x, filter_depth, layers[0], init_stride=1)
    for i in range(len(layers[1:])):
        x = res_blk(x, (2 ** (i + 1)) * filter_depth, layers[i + 1], init_stride
=2)

    x = AveragePooling2D(padding="same")(x)
    x = Flatten()(x)
    x = Dropout(0.5)(x)
    x = Dense(classes, activation="softmax", kernel_initializer="he_normal")(x)
    model = Model(
        inputs=input, outputs=x, name=("ResNet-" + str(2 * np.sum(layers) + 2))
    )

    return model
```

```
import struct as st
import numpy as np
import pickle as pkl


class Parser(object):
    def __init__(self, dataset, type):
        self.type = type  # dataset type
        self.dataset = dataset  # dataset

    def parse(self):
        self.images_set = []
        self.labels_set = []
        if (
            self.type == "MNIST"
        ):  # dataset follows a dictionary construct based on images and labels
            for k, v in self.dataset.items():
                if "images" in k:
                    with open(v, "rb") as f:
                        magic, num_imgs, num_rows, num_cols = st.unpack(
                            ">IIII", f.read(16)
                        )
                        self.images_set.append(
                            np.fromfile(f, dtype=np.dtype(np.ubyte))
                            .newbyteorder(">")
                            .reshape(num_imgs, num_rows, num_cols, 1)
                        )
                elif "labels" in k:
                    with open(v, "rb") as f:
                        magic, num_items = st.unpack(">II", f.read(8))
                        self.labels_set.append(
                            np.fromfile(f, dtype=np.dtype(np.uint8)).newbyteorde
r(">")
                        )
        elif "CIFAR" in self.type:
            # dataset follows a dictionary construct based on training and test
            # https://mattpetersen.github.io/load-cifar10-with-numpy
            labels = b"fine_labels" if ("100" in self.type) else b"labels"
            for k, v in self.dataset.items():
                batch_dict = self.__unpickle(self.dataset[k])
                self.images_set.append(
                    np.transpose(
                        batch_dict[b"data"].reshape(
                            len(batch_dict[b"data"]), 3, 32, 32
                        ),
                        (0, 2, 3, 1),
                    )
                    / 255.0
                )
                self.labels_set.append(batch_dict[labels])

        return (
            (
                np.concatenate([*self.images_set[:-1]]),
                np.array(self.labels_set[:-1]).flatten(),
            ),
            (self.images_set[-1], np.array(self.labels_set[-1]).flatten()),
        )

    # https://www.cs.toronto.edu/~kriz/cifar.html
    def __unpickle(self, file):
        with open(file, "rb") as fo:
```

```
            dict = pkl.load(fo, encoding="bytes")
        return dict
```

```
compile:
	black cifar_class.py resnet.py darse.py
	flake8 --ignore=E,W cifar_class.py resnet.py darse.py
	python3 cifar_class.py

pdf:
	a2ps cifar_class.py -o cifar_class.ps --pro=color
	a2ps resnet.py -o resnet.ps --pro=color
	a2ps darse.py -o darse.ps --pro=color
	a2ps Makefile -o Makefile.ps --pro=color
	ps2pdf cifar_class.ps
	ps2pdf resnet.ps
	ps2pdf darse.ps
	ps2pdf Makefile.ps
	gs -dBATCH -dNOPAUSE -q -sDEVICE=pdfwrite -sOutputFile=AR_CIFAR.pdf Assi
gnment4_CIFAR.pdf cifar_class.pdf resnet.pdf darse.pdf Makefile.pdf

clean:
	rm *.ps* *.pdf
```