ztl2103_coms6998_a3

November 3, 2020

1 COMS 6998 - Practical Deep Learning System Performance

1.1 Assignment 2

- **Name**: Zach Lawless
- **UNI**: ztl2103

1.1.1 Problem 1: *Adaptive Learning Rate Methods, CIFAR-10* (20 points)

Q1: (6 points) Below are the weight update equations explained as well as a description of the hyperparameters for each of the five adaptive learning rate methods in the problem statement. Let η represent the learning rate for all five learning rate methods.

(Note: Referenced this blog post by Sebastian Ruder.)

– AdaGrad – Because AdaGrad uses a different learning rate per model parameter θ_i at time t, first we must show AdaGrad's per-parameter update, which will then be vectorized. Letting $g_{t,i}$ represent the partial derivative of model parameter θ_i at time step t:

$$g_{t,i} = \nabla_\theta J(\theta_{t,i})$$

Where J represents the objective function we are trying to optimize over.

The Stochastic Gradient Descent for each model parameter θ_i at each time step t now becomes:

$$\theta_{t+1,i} = \theta_{t,i} - \eta g_{t,i}$$

Each update, AdaGrad changes the learning rate η for each model parameter based on the past gradients for each model parameter. Let G_t be an dxd matrix where each diagonal element i, i is the sum of the squares of the gradients w.r.t. θ_i up to time step t

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} g_{t,i}$$

Here, ϵ is a smoothing parameter so that we don't incur division by zero errors.

Because G_t contains the sum of squares of the gradients w.r.t θ on the diagonal, the implementation can be vectorized by taking the matrix vector product $\odot$ between G_t and g_t:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

– **RMSProp** – RMSProp is identical to the first update vector of AdaDelta below (using $\gamma = 0.9$):

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

The parameter update formula is thus:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g_t]^2 + \epsilon}} g_t$$

– **RMSProp + Nesterov** – Adam is essentially RMSProp with momentum. RMSProp + Nesterov momentum (also known as Nadam) takes it a step further to use Nesterov which is proven to be better than standard momentum.

The momentum update rule is:

$$g_t = \nabla_{\theta_t} J(\theta_t)$$

$$m_t = \gamma m_{t-1} + \eta g_t$$

$$\theta_{t+1} = \theta_t - m_t$$

In the above series of formulas, J is the objective funtion, γ is the momentum decay term, and η is the step size. We can combine the second into the third:

$$\theta_{t+1} = \theta_t - (\gamma m_{t-1} + \eta g_t)$$

This shows that momentum involves taking a step in the direction of the previous momentum vector as well as a step in the direction of the current gradient.

RMSProp + Nesterov allows us to take a more accurate step in the gradient direction by updating the parameters with the momentum step before computing the gradient. We thus only need to modify the gradient g_t to arrive at RMSProp + Nesterov:

$$g_t = \nabla_{\theta_t} J(\theta_t - \gamma m_{t-1})$$

$$m_t = \gamma m_{t-1} + \eta g_t$$

$$\theta_{t+1} = \theta_t - m_t$$

One note from above is that we are actually apply momentum twice; once in the gradient calculation as well as in the parameter update. One way to correct for this is to apply the momentum directly to the current parameter updates as opposed to the gradient.

$$g_t = \nabla_{\theta_t} J(\theta_t)$$

$$m_t = \gamma m_{t-1} + \eta g_t$$

$$\theta_{t+1} = \theta_t - (\gamma m_{t-1} + \eta g_t)$$

In order to add Nesterov momentum, we can thus similarly replace the previous momentum vector with the current momentum vector.

Using the Adam update formulas from below:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Plugging m_t and $\hat{m}_t$ into the parameter update equation yields:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \left(\frac{\beta_1 m_{t-1}}{1 - \beta_1^t} + \frac{(1 - \beta_1) g_t}{1 - \beta_1^t}\right)$$

We are still using the previous momentum vector, so to add Nesterov momentum, we simply use the current momentum vector. Simplifying and replacing $\frac{\beta_1 m_{t-1}}{1-\beta_1^t} = \hat{m}_{t-1}$, we get the final RMSProp + Nesterov update formula:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \left(\beta_1 \hat{m}_t + \frac{(1 - \beta_1) g_t}{1 - \beta_1^t}\right)$$

– AdaDelta – AdaDelta is similar to AdaGrad but slightly different in the sense that it restricts the accumulation of all past gradients to a fixed window w.

Instead of inefficiently storing w previous squared gradients, the sum of gradients is recursively defined as a decaying average of all past squared gradients. The running average $E[g^2]_t$ at time t thus depends only on the previous average and current gradient.

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma) g_t^2$$

This is the running average (E) of the squared gradients (g^2) at time (t) with momentum (γ).

We can simplify the paramater update.

$$\triangle\theta_t = -\eta \cdot g_t$$
$$\theta_{t+1} = \theta_t + \triangle\theta_t$$

Using the AdaGrad derived parameter update formula from above and replacing G_t with $E[g^2]_t$.

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

The denominator of the above formula is simply the Root Mean Squared error and can be rewritten.

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t$$

The units from the above formula do not match. To fix this, we define another exponentially decaying average not of the gradients but of the parameter updates.

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta^2$$

The root mean square error of the parameter updates is thus:

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}$$

Because the above is unknown, we can approximate it with the RMS of parameter updates until the previous time step. Replacing learning rate η with $RMS[\Delta\theta]_{t-1}$ yields:

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

– Adam – Adam also computes adaptive learning rates per model parameters. Adam stores the exponentially decaying average of past squared gradients v_t similarly to AdaDelta and RMSProp, but also stores the past exponentially decaying gradients m_t, similar to momentum. We compute the decaying averages of past and past squared gradients $m_t% and v_t respectively as follows:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)g_t^2$$

m_t and v_t are estimates of the first and second moment (mean and variance) of the gradients, respectively.

When m_t and v_t are initialized to vectors of 0s, they bias towards 0, especially in the initial time steps. In order to account for this, we take bias-corrected estimates as follows:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Using these in the same parameter update formula as AdaDelta and RMSProp yields the final Adam update rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

AdaDelta and Adam are different from RMSProp in the sense that both of them use momentum whereas RMSProp does not.

Q2: (5 points) Import the necessary libraries.

[ ]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import cifar10
from tensorflow.keras import layers, Model
from tensorflow.keras.utils import to_categorical

tf.__version__
```

[ ]:
```
'2.3.0'
```

Validate that the GPU is available for training.

[ ]:
```
tf.config.list_physical_devices('GPU')
```

[ ]:
```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

Load CIFAR-10 dataset and preprocess.

[ ]:
```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Set numeric type to float32 from uint8
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# Normalize value to [0, 1]
x_train /= 255
x_test /= 255

# flatten x matrices
x_train = x_train.reshape(x_train.shape[0], -1)
x_test = x_test.reshape(x_test.shape[0], -1)
```

```
# expand y targets
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

print(x_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 4s 0us/step
(50000, 3072) (50000, 10)
(10000, 3072) (10000, 10)
```

Create helper function that returns model with correct architecture for questions 2 and 3.

[ ]:
```
def create_model(model_name, optimizer, dropout=False):
  """ Return network for training. """

  # define input
  inputs = layers.Input(shape=(3072,), dtype='float32',␣
↪name=f'input_{model_name}')

  # add fully connected with dropout if needed
  if dropout:
    x = layers.Dropout(0.2, name=f'input_dropout_{model_name}')(inputs)
    for i in range(2):
      x = layers.Dense(1000,
                       activation='relu',
                       kernel_regularizer='l2',
                       bias_regularizer='l2',
                       name=f'dense_{i}_{model_name}')(x)
      x = layers.Dropout(0.5, name=f'dropout_{i}_{model_name}')(x)
  else:
      x = layers.Dense(1000,
                       activation='relu',
                       kernel_regularizer='l2',
                       bias_regularizer='l2',
                       name=f'dense_0_{model_name}')(inputs)
      x = layers.Dense(1000,
                       activation='relu',
                       kernel_regularizer='l2',
                       bias_regularizer='l2',
                       name=f'dense_1_{model_name}')(x)

  outputs = layers.Dense(10,
                         activation='softmax',
                         name=f'output_{model_name}')(x)
```

```
  model = Model(inputs=inputs, outputs=outputs, name=model_name)

  # compile
  if optimizer == 'adagrad':
    opt = keras.optimizers.Adagrad(name=f'adagrad_{model_name}')
  elif optimizer == 'rmsprop':
    opt = keras.optimizers.RMSprop(name=f'rmsprop_{model_name}')
  elif optimizer == 'nadam':
    opt = keras.optimizers.Nadam(name=f'nadam_{model_name}')
  elif optimizer == 'adadelta':
    opt = keras.optimizers.Adadelta(name=f'adadelta_{model_name}')
  elif optimizer == 'adam':
    opt = keras.optimizers.Adam(name=f'adam_{model_name}')
  else:
    raise NotImplementedError

  model.compile(loss='categorical_crossentropy',
                optimizer=opt,
                metrics=['acc'])

  # return
  return model
```

Loop through the five optimizer options and train.

```
[ ]: from time import time
```

```
[ ]: OPTIMIZERS = ['adagrad', 'rmsprop', 'nadam', 'adadelta', 'adam']
     METRICS = dict()
     for opt in OPTIMIZERS:
       print(f'training {opt} without dropout...')
       model = create_model(model_name = opt, optimizer=opt, dropout=False)
       t1 = time()
       history = model.fit(x=x_train,
                           y=y_train,
                           epochs=200,
                           batch_size=128,
                           verbose=0)
       t2 = time()
       METRICS[opt] = {
           'history': history,
           'training_time': t2 - t1,
           'test_acc': model.evaluate(x_test, y_test, return_dict=True,␣
      ↪verbose=0)['acc']
       }
```

```
training adagrad without dropout…
training rmsprop without dropout…
```

```
training nadam without dropout…
training adadelta without dropout…
training adam without dropout…
```

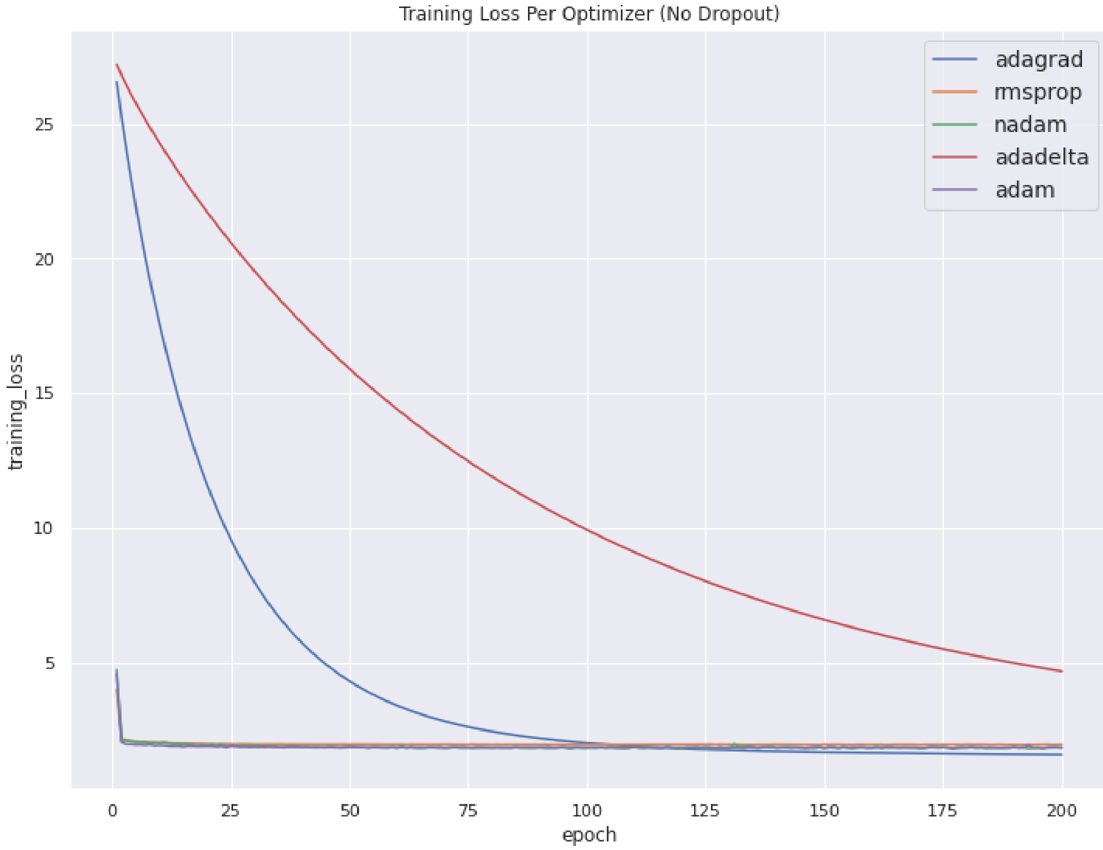
Plot the training loss per epoch for each optimization model.

[ ]:
```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
sns.set_theme(style="darkgrid")
```

[ ]:
```
DATA = {
    'optimizer': [],
    'epoch': [],
    'training_loss': []
}

for opt in METRICS.keys():
  opt_hist = METRICS[opt]['history']
  opt_loss = opt_hist.history['loss']
  DATA['optimizer'] += [opt] * 200
  DATA['epoch'] += [i+1 for i in range(200)]
  DATA['training_loss'] += opt_loss
  print(f'{opt}\n\tMinimum Training Loss: {min(opt_loss)}\n\tFinal Training␣
 ↪Loss: {opt_loss[-1]}')

df = pd.DataFrame(data=DATA)

fig, ax = plt.subplots(figsize=(12, 9))
sns.lineplot(x='epoch', y='training_loss', hue='optimizer', data=df)
plt.legend(fontsize=14)
plt.title('Training Loss Per Optimizer (No Dropout)')
```

```
adagrad
        Minimum Training Loss: 1.5973317623138428
        Final Training Loss: 1.5973317623138428
rmsprop
        Minimum Training Loss: 1.9577230215072632
        Final Training Loss: 1.9685896635055542
nadam
        Minimum Training Loss: 1.817350149154663
        Final Training Loss: 1.848476529121399
adadelta
        Minimum Training Loss: 4.688837051391602
        Final Training Loss: 4.688837051391602
adam
        Minimum Training Loss: 1.8387531042099
        Final Training Loss: 1.8734663724899292
```

[ ]: Text(0.5, 1.0, 'Training Loss Per Optimizer (No Dropout)')



Per the line plot and diagnostics printed above, the AdaGrad optimizer achieved the smallest loss in the 200 epochs as well as finished with the smallest loss at the end of training.

Q3: (5 points) Repeating the exercise but with dropout introduced to the networks.

[ ]:
```
OPTIMIZERS = ['adagrad', 'rmsprop', 'nadam', 'adadelta', 'adam']
METRICS_DROPOUT = dict()
for opt in OPTIMIZERS:
  print(f'training {opt} with dropout...')
  model = create_model(model_name = opt, optimizer=opt, dropout=True)
  t1 = time()
  history = model.fit(x=x_train,
                      y=y_train,
                      epochs=200,
                      batch_size=128,
                      verbose=0)
  t2 = time()
  METRICS_DROPOUT[opt] = {
```

```
        'history': history,
        'training_time': t2 - t1,
        'test_acc': model.evaluate(x_test, y_test, return_dict=True,␣
 ↪verbose=0)['acc']
   }
```

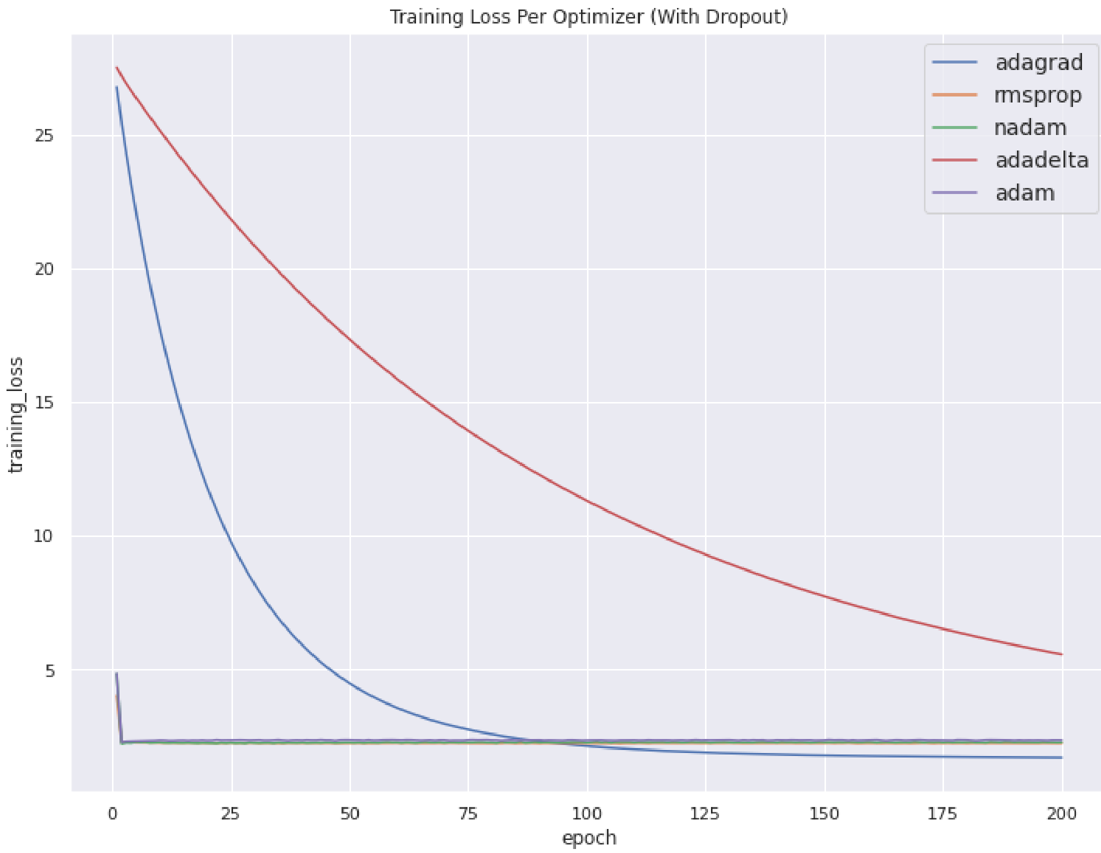
```
training adagrad with dropout…
training rmsprop with dropout…
training nadam with dropout…
training adadelta with dropout…
training adam with dropout…
```

[ ]:
```
DATA = {
    'optimizer': [],
    'epoch': [],
    'training_loss': []
}

for opt in METRICS_DROPOUT.keys():
  opt_hist = METRICS_DROPOUT[opt]['history']
  opt_loss = opt_hist.history['loss']
  DATA['optimizer'] += [opt] * 200
  DATA['epoch'] += [i+1 for i in range(200)]
  DATA['training_loss'] += opt_loss
  print(f'{opt}\n\tMinimum Training Loss: {min(opt_loss)}\n\tFinal Training␣
 ↪Loss: {opt_loss[-1]}')

df_dropout = pd.DataFrame(data=DATA)

fig, ax = plt.subplots(figsize=(12, 9))
sns.lineplot(x='epoch', y='training_loss', hue='optimizer', data=df_dropout)
plt.legend(fontsize=14)
plt.title('Training Loss Per Optimizer (With Dropout)')
```

```
adagrad
        Minimum Training Loss: 1.7147144079208374
        Final Training Loss: 1.7147144079208374
rmsprop
        Minimum Training Loss: 2.243259906768799
        Final Training Loss: 2.252845525741577
nadam
        Minimum Training Loss: 2.2300102710723877
        Final Training Loss: 2.2887513637542725
adadelta
        Minimum Training Loss: 5.573465824127197
        Final Training Loss: 5.573465824127197
adam
```

```
Minimum Training Loss: 2.293626546859741
Final Training Loss: 2.3687984943389893
```

[ ]: Text(0.5, 1.0, 'Training Loss Per Optimizer (With Dropout)')



Compare non-dropout models to dropout models for training time and loss.

[ ]:
```
for opt in OPTIMIZERS:
  print(f'--- {opt} ---')

  non_dropout_opt_metrics = METRICS[opt]
  dropout_opt_metrics = METRICS_DROPOUT[opt]
  print('\tTraining Time:')
  print(f"\t\tNon Dropout: {non_dropout_opt_metrics['training_time']}")
  print(f"\t\tDropout: {dropout_opt_metrics['training_time']}")
  print('\tTraining Loss:')
  print(f"\t\tNon Dropout: {min(non_dropout_opt_metrics['history'].
↪history['loss'])}")
  print(f"\t\tDropout: {min(dropout_opt_metrics['history'].history['loss'])}")
```

```
--- adagrad ---
```

```
        Training Time:
                Non Dropout: 415.31490755081177
                Dropout: 444.38312220573425
        Training Loss:
                Non Dropout: 1.5973317623138428
                Dropout: 1.7147144079208374
--- rmsprop ---
        Training Time:
                Non Dropout: 554.7393670082092
                Dropout: 584.0763900279999
        Training Loss:
                Non Dropout: 1.9577230215072632
                Dropout: 2.243259906768799
--- nadam ---
        Training Time:
                Non Dropout: 788.6662545204163
                Dropout: 812.9193739891052
        Training Loss:
                Non Dropout: 1.817350149154663
                Dropout: 2.2300102710723877
--- adadelta ---
        Training Time:
                Non Dropout: 461.5627610683441
                Dropout: 484.38758873939514
        Training Loss:
                Non Dropout: 4.688837051391602
                Dropout: 5.573465824127197
--- adam ---
        Training Time:
                Non Dropout: 423.1025047302246
                Dropout: 448.29078125953674
        Training Loss:
                Non Dropout: 1.8387531042099
                Dropout: 2.293626546859741
```

Q4: (4 points) Comparing the test accuracy for all optimizers and their respective dropout and non-dropout models.

[ ]:
```
for opt in OPTIMIZERS:
  print(f'--- {opt} ---')

  non_dropout_opt_metrics = METRICS[opt]
  dropout_opt_metrics = METRICS_DROPOUT[opt]
  print('\tTest Accuracy:')
  print(f"\t\tNon Dropout: {non_dropout_opt_metrics['test_acc']}")
  print(f"\t\tDropout: {dropout_opt_metrics['test_acc']}")
```

```
--- adagrad ---
        Test Accuracy:
                Non Dropout: 0.5102999806404114
                Dropout: 0.4805000126361847
--- rmsprop ---
        Test Accuracy:
                Non Dropout: 0.3312000036239624
                Dropout: 0.25529998540878296
--- nadam ---
        Test Accuracy:
                Non Dropout: 0.39750000834465027
                Dropout: 0.2567000091075897
--- adadelta ---
        Test Accuracy:
                Non Dropout: 0.4507000148296356
                Dropout: 0.40369999408721924
--- adam ---
        Test Accuracy:
                Non Dropout: 0.41760000586509705
                Dropout: 0.28119999170303345
```

1.1.2 Problem 2: *TF 2.0, tensorflow.distribute.strategy, Strong and Weak Scaling* (0 points)

Not doable due to GCP limits.

1.1.3 Problem 3: *Convolutional Neural Networks Architectures* (30 points)

Q1: (5 points) Below is a table depicting the inputs, outputs, and number of parameters for each layer in AlexNet (note, bias is included):

Layer	Input Size	Layer Description	Number of Parameters	Output Size
Input	3 x 27 x 27			3 x 27 x 27
Convolution	3 x 27 x 27	11 x 11 filter, 96 filters, stride 4	(11 x 11 x 3 + 1) x 96 = 34944	96 x 55 x 55
Max Pooling	96 x 55 x 55	3 x 3 filter, stride 2		96 x 27 x 27
Normalizing	96 x 27 x 27			96 x 27 x 27
Convolution	96 x 27 x 27	5 x 5 filter, 256 filters, stride 1, padding 2	(5 x 5 x 96 + 1) x 256 = 614656	256 x 27 x 27
Max Pooling	256 x 27 x 27	3 x 3 filter, stride 2		256 x 13 x 13
Normalizing	256 x 13 x 13			256 x 13 x 13
Convolution	256 x 13 x 13	3 x 3 filter, 384 filters, stride 1, padding 1	(3 x 3 x 256 + 1) x 384 = 885120	384 x 13 x 13

Layer	Input Size	Layer Description	Number of Parameters	Output Size
Convolution	384 x 13 x 13	3 x 3 filters, 384 filters, stride 1, padding 1	(3 x 3 x 384 + 1) x 384 = 1327488	384 x 13 x 13
Convolution	384 x 13 x 13	3 x 3 filters, 256 filters, stride 1, padding 1	(3 x 3 x 384 + 1) x 256 = 884992	256 x 13 x 13
Max Pooling	256 x 13 x 13	3 x 3 filter, stride 2		256 x 6 x 6
Dense	256 x 6 x 6	4096 units	256 x 6 x 6 x 4096 = 37748736	4096
Dense	4096	4096 units	4096 x 4096 = 16777216	4096
Dense	4096	1000 units	4096 x 1000 = 4096000	1000

Summing the Number of Parameters columns shows that AlexNet in total contains 62369152 learning parameters.

Q2: (6 points) The completed VGG19 memory and weights table follows.

Layer	Number of Activations (Memory)	Parameters (Compute)
Input	224 x 224 x 3 = 150K	0
CONV3-64	224 x 224 x 64 = 3.2M	(3 x 3 x 3) x 64 = 1728
CONV3-64	224 x 224 x 64 = 3.2M	(3 x 3 x 64) x 64 = 36864
POOL2	112 x 112 x 64 = 800K	0
CONV3-128	112 x 112 x 128 = 1.6M	(3 x 3 x 64) x 128 = 73728
CONV3-128	112 x 112 x 128 = 1.6M	(3 x 3 x 128) x 128 = 147456
POOL2	56 x 56 x 128 = 400K	0
CONV3-256	56 x 56 x 256 = 800K	(3 x 3 x 128) x 256 = 294912
CONV3-256	56 x 56 x 256 = 800K	(3 x 3 x 256) x 256 = 589824
CONV3-256	56 x 56 x 256 = 800K	(3 x 3 x 256) x 256 = 589824
CONV3-256	56 x 56 x 256 = 800K	(3 x 3 x 256) x 256 = 589824
POOL2	28 x 28 x 256 = 200K	0
CONV3-512	28 x 28 x 512 = 400K	(3 x 3 x 256) x 512 = 1179648
CONV3-512	28 x 28 x 512 = 400K	(3 x 3 x 512) x 512 = 2359296
CONV3-512	28 x 28 x 512 = 400K	(3 x 3 x 512) x 512 = 2359296
CONV3-512	28 x 28 x 512 = 400K	(3 x 3 x 512) x 512 = 2359296
POOL2	14 x 14 x 512 = 100K	0
CONV3-512	14 x 14 x 512 = 100K	(3 x 3 x 512) x 512 = 2359296
CONV3-512	14 x 14 x 512 = 100K	(3 x 3 x 512) x 512 = 2359296
CONV3-512	14 x 14 x 512 = 100K	(3 x 3 x 512) x 512 = 2359296
CONV3-512	14 x 14 x 512 = 100K	(3 x 3 x 512) x 512 = 2359296
POOL2	7 x 7 x 512 = 25K	0
FC	4096	(7 x 7 x 512) x 4096 = 102760448
FC	4096	4096 x 4096 = 16777216
FC	1000	4096 x 1000 = 4096000
TOTAL	16M	143652544

Q3: (4 points) Knowing that the output of an convolution activation field is $L - F + 1$ where L and F are the length of the input image (assuming square) and convolution filter size (assuming square as well) respectively, we can start the proof by induction.

The first of N successive stacks of convolutions on an FxF filter results in an output activation map of:

$$L - F + 1$$

Repeating the convolution on this output yields:

$$(L - F + 1) - F + 1 = L - 2F + 2$$

Repeating a third time results in:

$$(L - 2F + 2) - F + 1 = L - 3F + 3$$

It is easy to see that stacking N convolution filters successively results in an output activation size $ L - NF + N $.

Now, looking at one convolution of filter size $ (NF - N + 1) $ yields:

$$L - (NF - N + 1) + 1 = L - NF + N$$

This is the same as above and the proof is correct.

Using the formula to calculate the output of 3 successive 5 x 5 filters results in an output size of:

$$L - 3x5 + 3 = L - 12$$

Using CIFAR-10 32x32 images as an example, the output size of 3 5x5 filters on this dataset would be a 20 x 20 feature map.

Q4:

a: (3 points) The general idea behind an inception module is to preserve local feature correlations of various sizes while also maintaining the sparsity that leads to reasonable training time and model size. Inception modules can be thought of as a dimensionality reduction technique within network while still allowing for state-of-the-art model performance.

b: (4 points) For the naive version and the dimensionality reduction version, we zero-pad each convolution so that they can be concatenated together.

For the naive Inception module, the output of the three convolution branches and one max pooling branch are $32x32x128$, $32x32x192$, $32x32x96$, and $32x32x256$ respectively. This results in a dimension of $32x32x672$ coming out of the concatenation layer.

As for the dimensionality reduction module, the output of the three convolution branches and one max pooling branch are $32x32x128$, $32x32x192$, $32x32x96$, and $32x32x64$ respectively. This results in a dimension of $32x32x480$ coming out of the concatenation layer.

c: (4 points) For the Naive architecture:

```
# First Convolution Block
(32 x 32) x (1 x 1 x 256) x 128 = 33,554,432 Multiplications

# Second Convolution Block
(32 x 32) x (3 x 3 x 256) x 192 = 452,984,832 Multiplications

# Third Convolution Block
(32 x 32) x (5 x 5 x 256) x 96 = 629,145,600 Multiplications

TOTAL = 33,554,432 + 452,984,832 + 629,145,600 = 1.11B Multiplications
```

For the Dimensionality Reduction architecture

```
# First Convolution Block
(32 x 32) x (1 x 1 x 256) x 128 = 33,554,432 Multiplications

# Second Convolution Block
(32 x 32) x (1 x 1 x 256) x 128 = 33,554,432 Multiplications
(32 x 32) x (3 x 3 x 128) x 192 = 226,492,416 Multiplications

# Third Convolution Block
(32 x 32) x (1 x 1 x 256) x 32 = 8,388,608 Multiplications
(32 x 32) x (5 x 5 x 32) x 96 = 78,643,200 Multiplications

# Max Pooling Block
(32 x 32) x (1 x 1 x 256) x 64 = 16,777,216 Multiplications

TOTAL = 33,554,432 + (33,554,432 + 226,492,416) + (8,388,608 + 78,643,200) + 16,777,216 = 397M
```

d: (4 points) The naive architecture still generates a lot of parameters despite the attempt to go "wide" instead of "deep". The dimensionality reduction architecture helps in this by first reducing the dimension with n smaller $1x1$ filters where n is less than the input dimension of the Inception module.

Based on the calculations from part c, the dimensionality reduction architecture reduces the number of multiplications needed in the naive architecture by a factor of roughly 3.

1.1.4 Problem 4: *Batch Augmentation, Cutout Regularization* (20 points)

Q1: (6 points) Cutout regularization has a few advantages over simple dropout when applied to computer vision. In general, convolutional layers have fewer parameters than dense layers, and thus don't need as much regularization. Also, the neighboring features (pixels) in images often contain the same or very similar information, and applying dropout on these images doesn't result in the averaging effect observed in dense layers. Cutout also has a pleasant property in that the regularization technique is applied at input and no where else in the network, which in turn requires zero model adaptation at inference. Appling cutout at the input cascades the cutout throughout the network's feature maps inherently as if dropout were being applied to the same features regardless of feature map, which is different than the random nature of dropout from layer to layer not continuing the feature masking from prior layers.

Below are two examples of images from CIFAR-10 before and after Cutout has been applied.

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.datasets import cifar10

(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Set numeric type to float32 from uint8
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# Normalize value to [0, 1]
x_train /= 255
x_test /= 255
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 4s 0us/step
```

[2]:
```
# Create a function to apply cutout to an image
def cutout(img, length, n_holes=1):
  """
  Args:
      img (np.ndarray): image of size (H, W, C).
  Returns:
      Tensor: Image with n_holes of dimension length x length cut out of it.
  """
  h = img.shape[0]
  w = img.shape[1]

  mask = np.ones((h, w), np.float32)

  for n in range(n_holes):
      y = np.random.randint(h)
      x = np.random.randint(w)

      y1 = np.clip(y - length // 2, 0, h)
      y2 = np.clip(y + length // 2, 0, h)
      x1 = np.clip(x - length // 2, 0, w)
      x2 = np.clip(x + length // 2, 0, w)

      mask[y1: y2, x1: x2] = 0.
```

```
    mask = np.expand_dims(mask, axis=2)
    mask = np.repeat(mask, repeats=3, axis=2)
    img = img * mask

    return img
```

[3]:
```
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(12, 12))
ax1.imshow(x_train[7])
ax2.imshow(cutout(x_train[7], length=12))
ax3.imshow(x_train[3])
ax4.imshow(cutout(x_train[3], length=12))
plt.show()
```

Q2: (4 points) Using this website as reference.

Create a helper function to build a `ResNet` layer.

[4]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, Conv2D, BatchNormalization,␣
 ↪Activation
from tensorflow.keras.layers import AveragePooling2D, Input, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import LearningRateScheduler,␣
 ↪ReduceLROnPlateau, EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.regularizers import l2
from tensorflow.keras import backend as K
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import cifar10
import numpy as np
import os
```

[5]:
```
def resnet_layer(inputs,
                 num_filters=16,
                 kernel_size=3,
                 strides=1,
                 activation='relu',
                 batch_normalization=True,
                 conv_first=True):
    """2D Convolution-Batch Normalization-Activation stack builder

    # Arguments
        inputs (tensor): input tensor from input image or previous layer
        num_filters (int): Conv2D number of filters
        kernel_size (int): Conv2D square kernel dimensions
        strides (int): Conv2D square stride dimensions
        activation (string): activation name
        batch_normalization (bool): whether to include batch normalization
        conv_first (bool): conv-bn-activation (True) or
            bn-activation-conv (False)

    # Returns
        x (tensor): tensor as input to the next layer
    """
    conv = Conv2D(num_filters,
                  kernel_size=kernel_size,
                  strides=strides,
                  padding='same',
                  kernel_initializer='he_normal',
                  kernel_regularizer=l2(1e-4))
```

```
    x = inputs
    if conv_first:
        x = conv(x)
        if batch_normalization:
            x = BatchNormalization()(x)
        if activation is not None:
            x = Activation(activation)(x)
    else:
        if batch_normalization:
            x = BatchNormalization()(x)
        if activation is not None:
            x = Activation(activation)(x)
        x = conv(x)
    return x
```

Create a function to build the `ResNet-44 V1` model.

[6]:
```
def resnet_v1(input_shape, depth, num_classes=10):
    """ResNet Version 1 Model builder [a]

    Stacks of 2 x (3 x 3) Conv2D-BN-ReLU
    Last ReLU is after the shortcut connection.
    At the beginning of each stage, the feature map size is halved (downsampled)
    by a convolutional layer with strides=2, while the number of filters is
    doubled. Within each stage, the layers have the same number filters and the
    same number of filters.
    Features maps sizes:
    stage 0: 32x32, 16
    stage 1: 16x16, 32
    stage 2:  8x8,  64
    The Number of parameters is approx the same as Table 6 of [a]:
    ResNet20 0.27M
    ResNet32 0.46M
    ResNet44 0.66M
    ResNet56 0.85M
    ResNet110 1.7M

    # Arguments
        input_shape (tensor): shape of input image tensor
        depth (int): number of core convolutional layers
        num_classes (int): number of classes (CIFAR10 has 10)

    # Returns
        model (Model): Keras model instance
    """
    if (depth - 2) % 6 != 0:
```

```python
        raise ValueError('depth should be 6n+2 (eg 20, 32, 44 in [a])')
    # Start model definition.
    num_filters = 16
    num_res_blocks = int((depth - 2) / 6)

    inputs = Input(shape=input_shape)
    x = resnet_layer(inputs=inputs)
    # Instantiate the stack of residual units
    for stack in range(3):
        for res_block in range(num_res_blocks):
            strides = 1
            if stack > 0 and res_block == 0:  # first layer but not first stack
                strides = 2  # downsample
            y = resnet_layer(inputs=x,
                             num_filters=num_filters,
                             strides=strides)
            y = resnet_layer(inputs=y,
                             num_filters=num_filters,
                             activation=None)
            if stack > 0 and res_block == 0:  # first layer but not first stack
                # linear projection residual shortcut connection to match
                # changed dims
                x = resnet_layer(inputs=x,
                                 num_filters=num_filters,
                                 kernel_size=1,
                                 strides=strides,
                                 activation=None,
                                 batch_normalization=False)
            x = keras.layers.add([x, y])
            x = Activation('relu')(x)
        num_filters *= 2

    # Add classifier on top.
    # v1 does not use BN after last shortcut connection-ReLU
    x = AveragePooling2D(pool_size=8)(x)
    y = Flatten()(x)
    outputs = Dense(num_classes,
                    activation='softmax',
                    kernel_initializer='he_normal')(y)

    # Instantiate model.
    model = Model(inputs=inputs, outputs=outputs)
    return model
```

Set up some global training parameters.

[7]:

```python
# Training parameters
batch_size = 64
epochs = 200
data_augmentation = True
num_classes = 10

# Subtracting pixel mean improves accuracy
subtract_pixel_mean = True

# which version of ResNet to use
version = 1

# Calculate n, depth for ResNet 44
n = 7
depth = n * 6 + 2

# Model name, depth and version
model_type = 'ResNet%dv%d' % (depth, version)
```

Reload CIFAR-10 and perform standard normalization and `to_categorical` operations.

[8]:

```python
# Load the CIFAR10 data.
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Input image dimensions.
input_shape = x_train.shape[1:]

# Normalize data.
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255

# If subtract pixel mean is enabled
if subtract_pixel_mean:
    x_train_mean = np.mean(x_train, axis=0)
    x_train -= x_train_mean
    x_test -= x_train_mean

print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
print('y_train shape:', y_train.shape)

# Convert class vectors to binary class matrices.
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
x_train shape: (50000, 32, 32, 3)
50000 train samples
```

```
10000 test samples
y_train shape: (50000, 1)
```

Set up learning rate decay with scheduler.

```
[9]: def lr_schedule(epoch):
    """Learning Rate Schedule

    Learning rate is scheduled to be reduced after 80, 120, 160, 180 epochs.
    Called automatically every epoch as part of callbacks during training.

    # Arguments
        epoch (int): The number of epochs

    # Returns
        lr (float32): learning rate
    """
    lr = 1e-3
    if epoch > 180:
        lr *= 0.5e-3
    elif epoch > 160:
        lr *= 1e-3
    elif epoch > 120:
        lr *= 1e-2
    elif epoch > 80:
        lr *= 1e-1
    print('Learning rate: ', lr)
    return lr
```

Build model.

```
[10]: model = resnet_v1(input_shape=input_shape, depth=depth)

model.compile(loss='categorical_crossentropy',
              optimizer=Adam(lr=lr_schedule(0)),
              metrics=['accuracy'])
model.summary()
print(model_type)
```

```
Learning rate:  0.001
Model: "functional_1"
________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
==================
input_1 (InputLayer)            [(None, 32, 32, 3)]  0
________________________________________________________________________________
__________________
```

```
conv2d (Conv2D)                 (None, 32, 32, 16)   448         input_1[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization (BatchNorma (None, 32, 32, 16)   64          conv2d[0][0]
--------------------------------------------------------------------------------
------------------
activation (Activation)         (None, 32, 32, 16)   0
batch_normalization[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_1 (Conv2D)               (None, 32, 32, 16)   2320
activation[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_1 (BatchNor (None, 32, 32, 16)   64          conv2d_1[0][0]
--------------------------------------------------------------------------------
------------------
activation_1 (Activation)       (None, 32, 32, 16)   0
batch_normalization_1[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_2 (Conv2D)               (None, 32, 32, 16)   2320
activation_1[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_2 (BatchNor (None, 32, 32, 16)   64          conv2d_2[0][0]
--------------------------------------------------------------------------------
------------------
add (Add)                       (None, 32, 32, 16)   0
activation[0][0]
batch_normalization_2[0][0]
--------------------------------------------------------------------------------
------------------
activation_2 (Activation)       (None, 32, 32, 16)   0           add[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_3 (Conv2D)               (None, 32, 32, 16)   2320
activation_2[0][0]
--------------------------------------------------------------------------------
------------------
batch_normalization_3 (BatchNor (None, 32, 32, 16)   64          conv2d_3[0][0]
--------------------------------------------------------------------------------
------------------
activation_3 (Activation)       (None, 32, 32, 16)   0
batch_normalization_3[0][0]
--------------------------------------------------------------------------------
------------------
conv2d_4 (Conv2D)               (None, 32, 32, 16)   2320
```

```
activation_3[0][0]
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 32, 32, 16)   64          conv2d_4[0][0]
__________________________________________________________________________________________________
add_1 (Add)                     (None, 32, 32, 16)   0
activation_2[0][0]
batch_normalization_4[0][0]
__________________________________________________________________________________________________
activation_4 (Activation)       (None, 32, 32, 16)   0           add_1[0][0]
__________________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 32, 32, 16)   2320
activation_4[0][0]
__________________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 32, 32, 16)   64          conv2d_5[0][0]
__________________________________________________________________________________________________
activation_5 (Activation)       (None, 32, 32, 16)   0
batch_normalization_5[0][0]
__________________________________________________________________________________________________
conv2d_6 (Conv2D)               (None, 32, 32, 16)   2320
activation_5[0][0]
__________________________________________________________________________________________________
batch_normalization_6 (BatchNor (None, 32, 32, 16)   64          conv2d_6[0][0]
__________________________________________________________________________________________________
add_2 (Add)                     (None, 32, 32, 16)   0
activation_4[0][0]
batch_normalization_6[0][0]
__________________________________________________________________________________________________
activation_6 (Activation)       (None, 32, 32, 16)   0           add_2[0][0]
__________________________________________________________________________________________________
conv2d_7 (Conv2D)               (None, 32, 32, 16)   2320
activation_6[0][0]
__________________________________________________________________________________________________
batch_normalization_7 (BatchNor (None, 32, 32, 16)   64          conv2d_7[0][0]
__________________________________________________________________________________________________
activation_7 (Activation)       (None, 32, 32, 16)   0
```

```
batch_normalization_7[0][0]
________________________________________________________________________________
__________________
conv2d_8 (Conv2D)               (None, 32, 32, 16)   2320
activation_7[0][0]
________________________________________________________________________________
__________________
batch_normalization_8 (BatchNor (None, 32, 32, 16)   64          conv2d_8[0][0]
________________________________________________________________________________
__________________
add_3 (Add)                     (None, 32, 32, 16)   0
activation_6[0][0]
batch_normalization_8[0][0]
________________________________________________________________________________
__________________
activation_8 (Activation)       (None, 32, 32, 16)   0           add_3[0][0]
________________________________________________________________________________
__________________
conv2d_9 (Conv2D)               (None, 32, 32, 16)   2320
activation_8[0][0]
________________________________________________________________________________
__________________
batch_normalization_9 (BatchNor (None, 32, 32, 16)   64          conv2d_9[0][0]
________________________________________________________________________________
__________________
activation_9 (Activation)       (None, 32, 32, 16)   0
batch_normalization_9[0][0]
________________________________________________________________________________
__________________
conv2d_10 (Conv2D)              (None, 32, 32, 16)   2320
activation_9[0][0]
________________________________________________________________________________
__________________
batch_normalization_10 (BatchNo (None, 32, 32, 16)   64          conv2d_10[0][0]
________________________________________________________________________________
__________________
add_4 (Add)                     (None, 32, 32, 16)   0
activation_8[0][0]
batch_normalization_10[0][0]
________________________________________________________________________________
__________________
activation_10 (Activation)      (None, 32, 32, 16)   0           add_4[0][0]
________________________________________________________________________________
__________________
conv2d_11 (Conv2D)              (None, 32, 32, 16)   2320
activation_10[0][0]
________________________________________________________________________________
__________________
```

```
batch_normalization_11 (BatchNo (None, 32, 32, 16)   64          conv2d_11[0][0]
________________________________________________________________________________
__________________
activation_11 (Activation)      (None, 32, 32, 16)   0
batch_normalization_11[0][0]
________________________________________________________________________________
__________________
conv2d_12 (Conv2D)              (None, 32, 32, 16)   2320
activation_11[0][0]
________________________________________________________________________________
__________________
batch_normalization_12 (BatchNo (None, 32, 32, 16)   64          conv2d_12[0][0]
________________________________________________________________________________
__________________
add_5 (Add)                     (None, 32, 32, 16)   0
activation_10[0][0]
batch_normalization_12[0][0]
________________________________________________________________________________
__________________
activation_12 (Activation)      (None, 32, 32, 16)   0           add_5[0][0]
________________________________________________________________________________
__________________
conv2d_13 (Conv2D)              (None, 32, 32, 16)   2320
activation_12[0][0]
________________________________________________________________________________
__________________
batch_normalization_13 (BatchNo (None, 32, 32, 16)   64          conv2d_13[0][0]
________________________________________________________________________________
__________________
activation_13 (Activation)      (None, 32, 32, 16)   0
batch_normalization_13[0][0]
________________________________________________________________________________
__________________
conv2d_14 (Conv2D)              (None, 32, 32, 16)   2320
activation_13[0][0]
________________________________________________________________________________
__________________
batch_normalization_14 (BatchNo (None, 32, 32, 16)   64          conv2d_14[0][0]
________________________________________________________________________________
__________________
add_6 (Add)                     (None, 32, 32, 16)   0
activation_12[0][0]
batch_normalization_14[0][0]
________________________________________________________________________________
__________________
activation_14 (Activation)      (None, 32, 32, 16)   0           add_6[0][0]
________________________________________________________________________________
__________________
```

```
conv2d_15 (Conv2D)              (None, 16, 16, 32)   4640
activation_14[0][0]

--------------------------------------------------------------------------------
------------------
batch_normalization_15 (BatchNo (None, 16, 16, 32)   128         conv2d_15[0][0]

--------------------------------------------------------------------------------
------------------
activation_15 (Activation)      (None, 16, 16, 32)   0
batch_normalization_15[0][0]

--------------------------------------------------------------------------------
------------------
conv2d_16 (Conv2D)              (None, 16, 16, 32)   9248
activation_15[0][0]

--------------------------------------------------------------------------------
------------------
conv2d_17 (Conv2D)              (None, 16, 16, 32)   544
activation_14[0][0]

--------------------------------------------------------------------------------
------------------
batch_normalization_16 (BatchNo (None, 16, 16, 32)   128         conv2d_16[0][0]

--------------------------------------------------------------------------------
------------------
add_7 (Add)                     (None, 16, 16, 32)   0           conv2d_17[0][0]
batch_normalization_16[0][0]

--------------------------------------------------------------------------------
------------------
activation_16 (Activation)      (None, 16, 16, 32)   0           add_7[0][0]

--------------------------------------------------------------------------------
------------------
conv2d_18 (Conv2D)              (None, 16, 16, 32)   9248
activation_16[0][0]

--------------------------------------------------------------------------------
------------------
batch_normalization_17 (BatchNo (None, 16, 16, 32)   128         conv2d_18[0][0]

--------------------------------------------------------------------------------
------------------
activation_17 (Activation)      (None, 16, 16, 32)   0
batch_normalization_17[0][0]

--------------------------------------------------------------------------------
------------------
conv2d_19 (Conv2D)              (None, 16, 16, 32)   9248
activation_17[0][0]

--------------------------------------------------------------------------------
------------------
batch_normalization_18 (BatchNo (None, 16, 16, 32)   128         conv2d_19[0][0]

--------------------------------------------------------------------------------
------------------
add_8 (Add)                     (None, 16, 16, 32)   0
```

```
activation_16[0][0]
batch_normalization_18[0][0]
________________________________________________________________________________
__________________
activation_18 (Activation)      (None, 16, 16, 32)   0           add_8[0][0]
________________________________________________________________________________
__________________
conv2d_20 (Conv2D)              (None, 16, 16, 32)   9248
activation_18[0][0]
________________________________________________________________________________
__________________
batch_normalization_19 (BatchNo (None, 16, 16, 32)   128         conv2d_20[0][0]
________________________________________________________________________________
__________________
activation_19 (Activation)      (None, 16, 16, 32)   0
batch_normalization_19[0][0]
________________________________________________________________________________
__________________
conv2d_21 (Conv2D)              (None, 16, 16, 32)   9248
activation_19[0][0]
________________________________________________________________________________
__________________
batch_normalization_20 (BatchNo (None, 16, 16, 32)   128         conv2d_21[0][0]
________________________________________________________________________________
__________________
add_9 (Add)                     (None, 16, 16, 32)   0
activation_18[0][0]
batch_normalization_20[0][0]
________________________________________________________________________________
__________________
activation_20 (Activation)      (None, 16, 16, 32)   0           add_9[0][0]
________________________________________________________________________________
__________________
conv2d_22 (Conv2D)              (None, 16, 16, 32)   9248
activation_20[0][0]
________________________________________________________________________________
__________________
batch_normalization_21 (BatchNo (None, 16, 16, 32)   128         conv2d_22[0][0]
________________________________________________________________________________
__________________
activation_21 (Activation)      (None, 16, 16, 32)   0
batch_normalization_21[0][0]
________________________________________________________________________________
__________________
conv2d_23 (Conv2D)              (None, 16, 16, 32)   9248
activation_21[0][0]
________________________________________________________________________________
__________________
```

```
batch_normalization_22 (BatchNo (None, 16, 16, 32)   128         conv2d_23[0][0]
________________________________________________________________________________
__________________
add_10 (Add)                    (None, 16, 16, 32)   0
activation_20[0][0]
batch_normalization_22[0][0]
________________________________________________________________________________
__________________
activation_22 (Activation)      (None, 16, 16, 32)   0           add_10[0][0]
________________________________________________________________________________
__________________
conv2d_24 (Conv2D)              (None, 16, 16, 32)   9248
activation_22[0][0]
________________________________________________________________________________
__________________
batch_normalization_23 (BatchNo (None, 16, 16, 32)   128         conv2d_24[0][0]
________________________________________________________________________________
__________________
activation_23 (Activation)      (None, 16, 16, 32)   0
batch_normalization_23[0][0]
________________________________________________________________________________
__________________
conv2d_25 (Conv2D)              (None, 16, 16, 32)   9248
activation_23[0][0]
________________________________________________________________________________
__________________
batch_normalization_24 (BatchNo (None, 16, 16, 32)   128         conv2d_25[0][0]
________________________________________________________________________________
__________________
add_11 (Add)                    (None, 16, 16, 32)   0
activation_22[0][0]
batch_normalization_24[0][0]
________________________________________________________________________________
__________________
activation_24 (Activation)      (None, 16, 16, 32)   0           add_11[0][0]
________________________________________________________________________________
__________________
conv2d_26 (Conv2D)              (None, 16, 16, 32)   9248
activation_24[0][0]
________________________________________________________________________________
__________________
batch_normalization_25 (BatchNo (None, 16, 16, 32)   128         conv2d_26[0][0]
________________________________________________________________________________
__________________
activation_25 (Activation)      (None, 16, 16, 32)   0
batch_normalization_25[0][0]
________________________________________________________________________________
__________________
```

```
conv2d_27 (Conv2D)              (None, 16, 16, 32)   9248
activation_25[0][0]
__________________________________________________________________________________
batch_normalization_26 (BatchNo (None, 16, 16, 32)   128         conv2d_27[0][0]
__________________________________________________________________________________
add_12 (Add)                    (None, 16, 16, 32)   0
activation_24[0][0]
batch_normalization_26[0][0]
__________________________________________________________________________________
activation_26 (Activation)      (None, 16, 16, 32)   0           add_12[0][0]
__________________________________________________________________________________
conv2d_28 (Conv2D)              (None, 16, 16, 32)   9248
activation_26[0][0]
__________________________________________________________________________________
batch_normalization_27 (BatchNo (None, 16, 16, 32)   128         conv2d_28[0][0]
__________________________________________________________________________________
activation_27 (Activation)      (None, 16, 16, 32)   0
batch_normalization_27[0][0]
__________________________________________________________________________________
conv2d_29 (Conv2D)              (None, 16, 16, 32)   9248
activation_27[0][0]
__________________________________________________________________________________
batch_normalization_28 (BatchNo (None, 16, 16, 32)   128         conv2d_29[0][0]
__________________________________________________________________________________
add_13 (Add)                    (None, 16, 16, 32)   0
activation_26[0][0]
batch_normalization_28[0][0]
__________________________________________________________________________________
activation_28 (Activation)      (None, 16, 16, 32)   0           add_13[0][0]
__________________________________________________________________________________
conv2d_30 (Conv2D)              (None, 8, 8, 64)     18496
activation_28[0][0]
__________________________________________________________________________________
batch_normalization_29 (BatchNo (None, 8, 8, 64)     256         conv2d_30[0][0]
__________________________________________________________________________________
```

```
activation_29 (Activation)      (None, 8, 8, 64)     0
batch_normalization_29[0][0]

--------------------------------------------------------------------------------
------------------
conv2d_31 (Conv2D)              (None, 8, 8, 64)     36928
activation_29[0][0]

--------------------------------------------------------------------------------
------------------
conv2d_32 (Conv2D)              (None, 8, 8, 64)     2112
activation_28[0][0]

--------------------------------------------------------------------------------
------------------
batch_normalization_30 (BatchNo (None, 8, 8, 64)     256         conv2d_31[0][0]

--------------------------------------------------------------------------------
------------------
add_14 (Add)                    (None, 8, 8, 64)     0           conv2d_32[0][0]
batch_normalization_30[0][0]

--------------------------------------------------------------------------------
------------------
activation_30 (Activation)      (None, 8, 8, 64)     0           add_14[0][0]

--------------------------------------------------------------------------------
------------------
conv2d_33 (Conv2D)              (None, 8, 8, 64)     36928
activation_30[0][0]

--------------------------------------------------------------------------------
------------------
batch_normalization_31 (BatchNo (None, 8, 8, 64)     256         conv2d_33[0][0]

--------------------------------------------------------------------------------
------------------
activation_31 (Activation)      (None, 8, 8, 64)     0
batch_normalization_31[0][0]

--------------------------------------------------------------------------------
------------------
conv2d_34 (Conv2D)              (None, 8, 8, 64)     36928
activation_31[0][0]

--------------------------------------------------------------------------------
------------------
batch_normalization_32 (BatchNo (None, 8, 8, 64)     256         conv2d_34[0][0]

--------------------------------------------------------------------------------
------------------
add_15 (Add)                    (None, 8, 8, 64)     0
activation_30[0][0]
batch_normalization_32[0][0]

--------------------------------------------------------------------------------
------------------
activation_32 (Activation)      (None, 8, 8, 64)     0           add_15[0][0]

--------------------------------------------------------------------------------
------------------
```

```
conv2d_35 (Conv2D)              (None, 8, 8, 64)     36928
activation_32[0][0]

--------------------------------------------------------------------------------
------------------
batch_normalization_33 (BatchNo (None, 8, 8, 64)     256         conv2d_35[0][0]

--------------------------------------------------------------------------------
------------------
activation_33 (Activation)      (None, 8, 8, 64)     0
batch_normalization_33[0][0]

--------------------------------------------------------------------------------
------------------
conv2d_36 (Conv2D)              (None, 8, 8, 64)     36928
activation_33[0][0]

--------------------------------------------------------------------------------
------------------
batch_normalization_34 (BatchNo (None, 8, 8, 64)     256         conv2d_36[0][0]

--------------------------------------------------------------------------------
------------------
add_16 (Add)                    (None, 8, 8, 64)     0
activation_32[0][0]
batch_normalization_34[0][0]

--------------------------------------------------------------------------------
------------------
activation_34 (Activation)      (None, 8, 8, 64)     0           add_16[0][0]

--------------------------------------------------------------------------------
------------------
conv2d_37 (Conv2D)              (None, 8, 8, 64)     36928
activation_34[0][0]

--------------------------------------------------------------------------------
------------------
batch_normalization_35 (BatchNo (None, 8, 8, 64)     256         conv2d_37[0][0]

--------------------------------------------------------------------------------
------------------
activation_35 (Activation)      (None, 8, 8, 64)     0
batch_normalization_35[0][0]

--------------------------------------------------------------------------------
------------------
conv2d_38 (Conv2D)              (None, 8, 8, 64)     36928
activation_35[0][0]

--------------------------------------------------------------------------------
------------------
batch_normalization_36 (BatchNo (None, 8, 8, 64)     256         conv2d_38[0][0]

--------------------------------------------------------------------------------
------------------
add_17 (Add)                    (None, 8, 8, 64)     0
activation_34[0][0]
batch_normalization_36[0][0]

--------------------------------------------------------------------------------
```

```
__________________
activation_36 (Activation)      (None, 8, 8, 64)     0           add_17[0][0]
________________________________________________________________________________
__________________
conv2d_39 (Conv2D)              (None, 8, 8, 64)     36928
activation_36[0][0]
________________________________________________________________________________
__________________
batch_normalization_37 (BatchNo (None, 8, 8, 64)     256         conv2d_39[0][0]
________________________________________________________________________________
__________________
activation_37 (Activation)      (None, 8, 8, 64)     0
batch_normalization_37[0][0]
________________________________________________________________________________
__________________
conv2d_40 (Conv2D)              (None, 8, 8, 64)     36928
activation_37[0][0]
________________________________________________________________________________
__________________
batch_normalization_38 (BatchNo (None, 8, 8, 64)     256         conv2d_40[0][0]
________________________________________________________________________________
__________________
add_18 (Add)                    (None, 8, 8, 64)     0
activation_36[0][0]
batch_normalization_38[0][0]
________________________________________________________________________________
__________________
activation_38 (Activation)      (None, 8, 8, 64)     0           add_18[0][0]
________________________________________________________________________________
__________________
conv2d_41 (Conv2D)              (None, 8, 8, 64)     36928
activation_38[0][0]
________________________________________________________________________________
__________________
batch_normalization_39 (BatchNo (None, 8, 8, 64)     256         conv2d_41[0][0]
________________________________________________________________________________
__________________
activation_39 (Activation)      (None, 8, 8, 64)     0
batch_normalization_39[0][0]
________________________________________________________________________________
__________________
conv2d_42 (Conv2D)              (None, 8, 8, 64)     36928
activation_39[0][0]
________________________________________________________________________________
__________________
batch_normalization_40 (BatchNo (None, 8, 8, 64)     256         conv2d_42[0][0]
________________________________________________________________________________
__________________
```

```
add_19 (Add)                    (None, 8, 8, 64)     0
activation_38[0][0]
batch_normalization_40[0][0]

--------------------------------------------------------------------------------
------------------
activation_40 (Activation)      (None, 8, 8, 64)     0           add_19[0][0]

--------------------------------------------------------------------------------
------------------
conv2d_43 (Conv2D)              (None, 8, 8, 64)     36928
activation_40[0][0]

--------------------------------------------------------------------------------
------------------
batch_normalization_41 (BatchNo (None, 8, 8, 64)     256         conv2d_43[0][0]

--------------------------------------------------------------------------------
------------------
activation_41 (Activation)      (None, 8, 8, 64)     0
batch_normalization_41[0][0]

--------------------------------------------------------------------------------
------------------
conv2d_44 (Conv2D)              (None, 8, 8, 64)     36928
activation_41[0][0]

--------------------------------------------------------------------------------
------------------
batch_normalization_42 (BatchNo (None, 8, 8, 64)     256         conv2d_44[0][0]

--------------------------------------------------------------------------------
------------------
add_20 (Add)                    (None, 8, 8, 64)     0
activation_40[0][0]
batch_normalization_42[0][0]

--------------------------------------------------------------------------------
------------------
activation_42 (Activation)      (None, 8, 8, 64)     0           add_20[0][0]

--------------------------------------------------------------------------------
------------------
average_pooling2d (AveragePooli (None, 1, 1, 64)     0
activation_42[0][0]

--------------------------------------------------------------------------------
------------------
flatten (Flatten)               (None, 64)           0
average_pooling2d[0][0]

--------------------------------------------------------------------------------
------------------
dense (Dense)                   (None, 10)           650         flatten[0][0]
================================================================================
==================
Total params: 665,994
Trainable params: 662,826
Non-trainable params: 3,168
```

```
--------------------------------------------------------------------------------
------------------
ResNet44v1
```

Set up callbacks for learning rate updating and early stopping during training.

[11]:
```
# Prepare callbacks for model saving and for learning rate adjustment.
lr_scheduler = LearningRateScheduler(lr_schedule)

lr_reducer = ReduceLROnPlateau(factor=np.sqrt(0.1),
                               cooldown=0,
                               patience=5,
                               min_lr=0.5e-6)

early_stopper = EarlyStopping(monitor='val_accuracy', patience=20)

callbacks = [lr_reducer, lr_scheduler, early_stopper]
```

Set up Data Generator for Image Augmentations.

[12]:
```
datagen = ImageDataGenerator(
        # set input mean to 0 over the dataset
        featurewise_center=False,
        # set each sample mean to 0
        samplewise_center=False,
        # divide inputs by std of dataset
        featurewise_std_normalization=False,
        # divide each input by its std
        samplewise_std_normalization=False,
        # apply ZCA whitening
        zca_whitening=False,
        # epsilon for ZCA whitening
        zca_epsilon=1e-06,
        # randomly rotate images in the range (deg 0 to 180)
        rotation_range=0,
        # randomly shift images horizontally
        width_shift_range=0.1,
        # randomly shift images vertically
        height_shift_range=0.1,
        # set range for random shear
        shear_range=0.,
        # set range for random zoom
        zoom_range=0.,
        # set range for random channel shifts
        channel_shift_range=0.,
        # set mode for filling points outside the input boundaries
        fill_mode='nearest',
        # value used for fill_mode = "constant"
```

```
        cval=0.,
        # randomly flip images
        horizontal_flip=True,
        # randomly flip images
        vertical_flip=False,
        # set rescaling factor (applied before any other transformation)
        rescale=None,
        # set function that will be applied on each input
        preprocessing_function=None,
        # image data format, either "channels_first" or "channels_last"
        data_format=None,
        # fraction of images reserved for validation (strictly between 0 and 1)
        validation_split=0.0)

datagen.fit(x_train)
```

Begin training.

[13]:
```
# Fit the model on the batches generated by datagen.flow().
model.fit_generator(datagen.flow(x_train, y_train, batch_size=batch_size),
                    validation_data=(x_test, y_test),
                    epochs=epochs, verbose=1, workers=4,
                    callbacks=callbacks)
```

```
WARNING:tensorflow:From <ipython-input-13-1b1c47dc6453>:5: Model.fit_generator
(from tensorflow.python.keras.engine.training) is deprecated and will be removed
in a future version.
Instructions for updating:
Please use Model.fit, which supports generators.
Learning rate:  0.001
Epoch 1/200
782/782 [==============================] - 39s 50ms/step - loss: 1.7936 -
accuracy: 0.4706 - val_loss: 2.0363 - val_accuracy: 0.4527
Learning rate:  0.001
Epoch 2/200
782/782 [==============================] - 38s 49ms/step - loss: 1.3394 -
accuracy: 0.6315 - val_loss: 1.3489 - val_accuracy: 0.6386
Learning rate:  0.001
Epoch 3/200
782/782 [==============================] - 37s 48ms/step - loss: 1.1306 -
accuracy: 0.7017 - val_loss: 1.1239 - val_accuracy: 0.7095
Learning rate:  0.001
Epoch 4/200
782/782 [==============================] - 39s 50ms/step - loss: 1.0167 -
accuracy: 0.7398 - val_loss: 1.5555 - val_accuracy: 0.6002
Learning rate:  0.001
Epoch 5/200
782/782 [==============================] - 38s 49ms/step - loss: 0.9377 -
```

```
accuracy: 0.7639 - val_loss: 1.3802 - val_accuracy: 0.6359
Learning rate:  0.001
Epoch 6/200
782/782 [==============================] - 38s 49ms/step - loss: 0.8871 -
accuracy: 0.7773 - val_loss: 1.1047 - val_accuracy: 0.7101
Learning rate:  0.001
Epoch 7/200
782/782 [==============================] - 38s 49ms/step - loss: 0.8399 -
accuracy: 0.7928 - val_loss: 1.8332 - val_accuracy: 0.5923
Learning rate:  0.001
Epoch 8/200
782/782 [==============================] - 38s 49ms/step - loss: 0.8042 -
accuracy: 0.8034 - val_loss: 1.0281 - val_accuracy: 0.7372
Learning rate:  0.001
Epoch 9/200
782/782 [==============================] - 39s 50ms/step - loss: 0.7714 -
accuracy: 0.8151 - val_loss: 1.2910 - val_accuracy: 0.6670
Learning rate:  0.001
Epoch 10/200
782/782 [==============================] - 40s 51ms/step - loss: 0.7416 -
accuracy: 0.8247 - val_loss: 1.2372 - val_accuracy: 0.6980
Learning rate:  0.001
Epoch 11/200
782/782 [==============================] - 39s 50ms/step - loss: 0.7256 -
accuracy: 0.8285 - val_loss: 1.5762 - val_accuracy: 0.6348
Learning rate:  0.001
Epoch 12/200
782/782 [==============================] - 39s 50ms/step - loss: 0.7048 -
accuracy: 0.8371 - val_loss: 1.0217 - val_accuracy: 0.7544
Learning rate:  0.001
Epoch 13/200
782/782 [==============================] - 39s 50ms/step - loss: 0.6811 -
accuracy: 0.8455 - val_loss: 0.9890 - val_accuracy: 0.7567
Learning rate:  0.001
Epoch 14/200
782/782 [==============================] - 40s 51ms/step - loss: 0.6636 -
accuracy: 0.8502 - val_loss: 0.9755 - val_accuracy: 0.7611
Learning rate:  0.001
Epoch 15/200
782/782 [==============================] - 39s 50ms/step - loss: 0.6534 -
accuracy: 0.8535 - val_loss: 0.9325 - val_accuracy: 0.7689
Learning rate:  0.001
Epoch 16/200
782/782 [==============================] - 39s 50ms/step - loss: 0.6413 -
accuracy: 0.8576 - val_loss: 1.0859 - val_accuracy: 0.7341
Learning rate:  0.001
Epoch 17/200
782/782 [==============================] - 39s 49ms/step - loss: 0.6272 -
```

```
accuracy: 0.8629 - val_loss: 0.9510 - val_accuracy: 0.7688
Learning rate:  0.001
Epoch 18/200
782/782 [==============================] - 39s 50ms/step - loss: 0.6211 -
accuracy: 0.8638 - val_loss: 1.0725 - val_accuracy: 0.7413
Learning rate:  0.001
Epoch 19/200
782/782 [==============================] - 39s 49ms/step - loss: 0.6045 -
accuracy: 0.8693 - val_loss: 0.8974 - val_accuracy: 0.7861
Learning rate:  0.001
Epoch 20/200
782/782 [==============================] - 39s 50ms/step - loss: 0.6016 -
accuracy: 0.8692 - val_loss: 1.3256 - val_accuracy: 0.6873
Learning rate:  0.001
Epoch 21/200
782/782 [==============================] - 39s 50ms/step - loss: 0.5910 -
accuracy: 0.8746 - val_loss: 0.9785 - val_accuracy: 0.7632
Learning rate:  0.001
Epoch 22/200
782/782 [==============================] - 40s 51ms/step - loss: 0.5840 -
accuracy: 0.8762 - val_loss: 0.6825 - val_accuracy: 0.8436
Learning rate:  0.001
Epoch 23/200
782/782 [==============================] - 39s 50ms/step - loss: 0.5756 -
accuracy: 0.8793 - val_loss: 0.7991 - val_accuracy: 0.8139
Learning rate:  0.001
Epoch 24/200
782/782 [==============================] - 39s 50ms/step - loss: 0.5693 -
accuracy: 0.8817 - val_loss: 0.8066 - val_accuracy: 0.8092
Learning rate:  0.001
Epoch 25/200
782/782 [==============================] - 39s 50ms/step - loss: 0.5601 -
accuracy: 0.8835 - val_loss: 0.8036 - val_accuracy: 0.8094
Learning rate:  0.001
Epoch 26/200
782/782 [==============================] - 40s 51ms/step - loss: 0.5569 -
accuracy: 0.8837 - val_loss: 0.7179 - val_accuracy: 0.8282
Learning rate:  0.001
Epoch 27/200
782/782 [==============================] - 39s 50ms/step - loss: 0.5512 -
accuracy: 0.8857 - val_loss: 0.6800 - val_accuracy: 0.8462
Learning rate:  0.001
Epoch 28/200
782/782 [==============================] - 39s 50ms/step - loss: 0.5481 -
accuracy: 0.8882 - val_loss: 0.7948 - val_accuracy: 0.8235
Learning rate:  0.001
Epoch 29/200
782/782 [==============================] - 39s 50ms/step - loss: 0.5388 -
```

```
accuracy: 0.8901 - val_loss: 0.7384 - val_accuracy: 0.8307
Learning rate:  0.001
Epoch 30/200
782/782 [==============================] - 39s 50ms/step - loss: 0.5345 -
accuracy: 0.8912 - val_loss: 0.8334 - val_accuracy: 0.8033
Learning rate:  0.001
Epoch 31/200
782/782 [==============================] - 39s 50ms/step - loss: 0.5318 -
accuracy: 0.8913 - val_loss: 0.7798 - val_accuracy: 0.8120
Learning rate:  0.001
Epoch 32/200
782/782 [==============================] - 39s 50ms/step - loss: 0.5233 -
accuracy: 0.8947 - val_loss: 0.7070 - val_accuracy: 0.8495
Learning rate:  0.001
Epoch 33/200
782/782 [==============================] - 39s 50ms/step - loss: 0.5227 -
accuracy: 0.8954 - val_loss: 0.6805 - val_accuracy: 0.8500
Learning rate:  0.001
Epoch 34/200
782/782 [==============================] - 39s 50ms/step - loss: 0.5196 -
accuracy: 0.8975 - val_loss: 0.7854 - val_accuracy: 0.8211
Learning rate:  0.001
Epoch 35/200
782/782 [==============================] - 39s 50ms/step - loss: 0.5152 -
accuracy: 0.8981 - val_loss: 0.6563 - val_accuracy: 0.8525
Learning rate:  0.001
Epoch 36/200
782/782 [==============================] - 39s 50ms/step - loss: 0.5065 -
accuracy: 0.9005 - val_loss: 0.7226 - val_accuracy: 0.8411
Learning rate:  0.001
Epoch 37/200
782/782 [==============================] - 39s 50ms/step - loss: 0.5090 -
accuracy: 0.8991 - val_loss: 0.7922 - val_accuracy: 0.8229
Learning rate:  0.001
Epoch 38/200
782/782 [==============================] - 39s 50ms/step - loss: 0.5076 -
accuracy: 0.8998 - val_loss: 0.8873 - val_accuracy: 0.8001
Learning rate:  0.001
Epoch 39/200
782/782 [==============================] - 39s 50ms/step - loss: 0.4982 -
accuracy: 0.9029 - val_loss: 1.0055 - val_accuracy: 0.7760
Learning rate:  0.001
Epoch 40/200
782/782 [==============================] - 39s 50ms/step - loss: 0.4995 -
accuracy: 0.9040 - val_loss: 0.6909 - val_accuracy: 0.8470
Learning rate:  0.001
Epoch 41/200
782/782 [==============================] - 39s 50ms/step - loss: 0.4951 -
```

```
accuracy: 0.9049 - val_loss: 0.6770 - val_accuracy: 0.8489
Learning rate:  0.001
Epoch 42/200
782/782 [==============================] - 39s 50ms/step - loss: 0.4940 -
accuracy: 0.9049 - val_loss: 0.8138 - val_accuracy: 0.8117
Learning rate:  0.001
Epoch 43/200
782/782 [==============================] - 39s 50ms/step - loss: 0.4892 -
accuracy: 0.9062 - val_loss: 0.6533 - val_accuracy: 0.8574
Learning rate:  0.001
Epoch 44/200
782/782 [==============================] - 39s 49ms/step - loss: 0.4851 -
accuracy: 0.9077 - val_loss: 0.7634 - val_accuracy: 0.8324
Learning rate:  0.001
Epoch 45/200
782/782 [==============================] - 39s 50ms/step - loss: 0.4860 -
accuracy: 0.9068 - val_loss: 0.7162 - val_accuracy: 0.8454
Learning rate:  0.001
Epoch 46/200
782/782 [==============================] - 39s 50ms/step - loss: 0.4749 -
accuracy: 0.9109 - val_loss: 0.6039 - val_accuracy: 0.8751
Learning rate:  0.001
Epoch 47/200
782/782 [==============================] - 39s 50ms/step - loss: 0.4789 -
accuracy: 0.9093 - val_loss: 0.8650 - val_accuracy: 0.8083
Learning rate:  0.001
Epoch 48/200
782/782 [==============================] - 39s 50ms/step - loss: 0.4760 -
accuracy: 0.9099 - val_loss: 0.6967 - val_accuracy: 0.8477
Learning rate:  0.001
Epoch 49/200
782/782 [==============================] - 39s 50ms/step - loss: 0.4731 -
accuracy: 0.9112 - val_loss: 0.8778 - val_accuracy: 0.7986
Learning rate:  0.001
Epoch 50/200
782/782 [==============================] - 39s 50ms/step - loss: 0.4729 -
accuracy: 0.9119 - val_loss: 0.7195 - val_accuracy: 0.8464
Learning rate:  0.001
Epoch 51/200
782/782 [==============================] - 39s 50ms/step - loss: 0.4673 -
accuracy: 0.9143 - val_loss: 0.7052 - val_accuracy: 0.8477
Learning rate:  0.001
Epoch 52/200
782/782 [==============================] - 39s 50ms/step - loss: 0.4719 -
accuracy: 0.9116 - val_loss: 0.8691 - val_accuracy: 0.8061
Learning rate:  0.001
Epoch 53/200
782/782 [==============================] - 39s 50ms/step - loss: 0.4670 -
```

```
accuracy: 0.9139 - val_loss: 0.6955 - val_accuracy: 0.8457
Learning rate:  0.001
Epoch 54/200
782/782 [==============================] - 39s 50ms/step - loss: 0.4608 -
accuracy: 0.9159 - val_loss: 0.7701 - val_accuracy: 0.8314
Learning rate:  0.001
Epoch 55/200
782/782 [==============================] - 40s 51ms/step - loss: 0.4641 -
accuracy: 0.9133 - val_loss: 0.6818 - val_accuracy: 0.8517
Learning rate:  0.001
Epoch 56/200
782/782 [==============================] - 39s 50ms/step - loss: 0.4621 -
accuracy: 0.9154 - val_loss: 0.8817 - val_accuracy: 0.8042
Learning rate:  0.001
Epoch 57/200
782/782 [==============================] - 40s 51ms/step - loss: 0.4568 -
accuracy: 0.9170 - val_loss: 0.8138 - val_accuracy: 0.8174
Learning rate:  0.001
Epoch 58/200
782/782 [==============================] - 39s 50ms/step - loss: 0.4546 -
accuracy: 0.9180 - val_loss: 0.6436 - val_accuracy: 0.8627
Learning rate:  0.001
Epoch 59/200
782/782 [==============================] - 39s 50ms/step - loss: 0.4578 -
accuracy: 0.9160 - val_loss: 0.7603 - val_accuracy: 0.8300
Learning rate:  0.001
Epoch 60/200
782/782 [==============================] - 39s 50ms/step - loss: 0.4539 -
accuracy: 0.9160 - val_loss: 0.7903 - val_accuracy: 0.8244
Learning rate:  0.001
Epoch 61/200
782/782 [==============================] - 37s 48ms/step - loss: 0.4580 -
accuracy: 0.9151 - val_loss: 1.0302 - val_accuracy: 0.7671
Learning rate:  0.001
Epoch 62/200
782/782 [==============================] - 38s 49ms/step - loss: 0.4521 -
accuracy: 0.9184 - val_loss: 0.7812 - val_accuracy: 0.8307
Learning rate:  0.001
Epoch 63/200
782/782 [==============================] - 39s 50ms/step - loss: 0.4558 -
accuracy: 0.9162 - val_loss: 0.6481 - val_accuracy: 0.8674
Learning rate:  0.001
Epoch 64/200
782/782 [==============================] - 38s 49ms/step - loss: 0.4496 -
accuracy: 0.9188 - val_loss: 0.8541 - val_accuracy: 0.8043
Learning rate:  0.001
Epoch 65/200
782/782 [==============================] - 39s 50ms/step - loss: 0.4481 -
```

```
accuracy: 0.9199 - val_loss: 0.7220 - val_accuracy: 0.8487
Learning rate:  0.001
Epoch 66/200
782/782 [==============================] - 38s 49ms/step - loss: 0.4439 -
accuracy: 0.9195 - val_loss: 0.7512 - val_accuracy: 0.8425
```

[13]:
```
<tensorflow.python.keras.callbacks.History at 0x7f45c46f0828>
```

Plotting the validation error vs number of epochs.

[5]:
```python
import seaborn as sns
sns.set_theme(style="darkgrid")
```

[15]:
```python
y = model.history.history.get('val_accuracy')
y = [1 - i for i in y]
x = [i+1 for i in range(len(y))]

fig, ax = plt.subplots(figsize=(12, 9))
sns.lineplot(x=x, y=y)
ax.set_xlabel('Epoch')
ax.set_ylabel('Validation Error')
ax.set_title('ResNet 44 CIFAR-10 Validation Error')
```

[15]:
```
Text(0.5, 1.0, 'ResNet 44 CIFAR-10 Validation Error')
```

Q3: (10 points)

[13]:
```
from time import time
```

Re-download CIFAR-10.

[14]:
```
# Training parameters
batch_size = 64
epochs = 100
num_classes = 10

# Load the CIFAR10 data.
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Input image dimensions.
input_shape = x_train.shape[1:]

# Normalize data.
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
```

```
# Convert class vectors to binary class matrices.
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

My custom data augmentation function.

[15]:
```
def my_data_augmentation(x_train, y_train, m):
  x_train_m = np.repeat(x_train, repeats=m, axis=0)
  y_train_m = np.repeat(y_train, repeats=m, axis=0)
  for i in range(x_train_m.shape[0]):
    x_train_m[i] = cutout(x_train_m[i], length=12)
  return x_train_m, y_train_m
```

Perform experiments.

Note: Due to Colab behaviour, I manually run these next few cells for various values of `m`, log the time to train in a markdown table, and persist the validation data to Google Drive to be loaded back in later.

[16]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

[17]:
```
def run_experiment(m):
  print(f'--- M={m} ---')

  # perform data augmentation
  print('performing data augmentation...')
  x_train_m, y_train_m = my_data_augmentation(x_train, y_train, m)
  datagen = ImageDataGenerator(
          # set input mean to 0 over the dataset
          featurewise_center=False,
          # set each sample mean to 0
          samplewise_center=False,
          # divide inputs by std of dataset
          featurewise_std_normalization=False,
          # divide each input by its std
          samplewise_std_normalization=False,
          # apply ZCA whitening
          zca_whitening=False,
          # epsilon for ZCA whitening
          zca_epsilon=1e-06,
          # randomly rotate images in the range (deg 0 to 180)
          rotation_range=0,
          # randomly shift images horizontally
          width_shift_range=0.1,
          # randomly shift images vertically
```

```
        height_shift_range=0.1,
        # set range for random shear
        shear_range=0.,
        # set range for random zoom
        zoom_range=0.,
        # set range for random channel shifts
        channel_shift_range=0.,
        # set mode for filling points outside the input boundaries
        fill_mode='nearest',
        # value used for fill_mode = "constant"
        cval=0.,
        # randomly flip images
        horizontal_flip=True,
        # randomly flip images
        vertical_flip=False,
        # set rescaling factor (applied before any other transformation)
        rescale=None,
        # set function that will be applied on each input
        preprocessing_function=None,
        # image data format, either "channels_first" or "channels_last"
        data_format=None,
        # fraction of images reserved for validation (strictly between 0 and␣
↪1)
        validation_split=0.0)
datagen.fit(x_train_m)

# set up model
print('setting up model...')
model = resnet_v1(input_shape=input_shape, depth=depth)
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(lr=.001),
              metrics=['accuracy'])
early_stopper = EarlyStopping(monitor='val_accuracy', patience=20)

# fit model
print('fitting model...')
t1 = time()
model.fit(datagen.flow(x_train_m, y_train_m, batch_size=batch_size),
          validation_data=(x_test, y_test),
          epochs=epochs, verbose=1, workers=4,
          callbacks=[early_stopper])
t2 = time()

return model.history, t2 - t1
```

[18]:
```
m = 16
```

[ ]:
```python
m_history, m_time = run_experiment(m)
```

```
--- M=16 ---
performing data augmentation…
```

NOTE: Run out of RAM when m=16 and using Google Colab.

[3]:
```python
import pandas as pd
```

[21]:
```python
val_acc = m_history.history['val_accuracy']

df_m = pd.DataFrame(data={'m': m, 'val_accuracy': val_acc})
df_m.head()
```

[21]:

	m	val_accuracy
0	8	0.7432
1	8	0.7436
2	8	0.7803
3	8	0.8596
4	8	0.8311

[22]:
```python
df_m['val_error'] = 1 - df_m['val_accuracy']
df_m.head()
```

[22]:

	m	val_accuracy	val_error
0	8	0.7432	0.2568
1	8	0.7436	0.2564
2	8	0.7803	0.2197
3	8	0.8596	0.1404
4	8	0.8311	0.1689

[23]:
```python
m_time
```

[23]:
```
10291.343039751053
```

[24]:
```python
df_m.to_csv(f'/content/gdrive/My Drive/COMS 6998/df{m}.csv', index=False)
```

Pulling saved files and plotting.

[4]:
```python
dfm2 = pd.read_csv('/content/gdrive/My Drive/COMS 6998/df2.csv')
dfm4 = pd.read_csv('/content/gdrive/My Drive/COMS 6998/df4.csv')
dfm8 = pd.read_csv('/content/gdrive/My Drive/COMS 6998/df8.csv')

dfm = pd.concat([dfm2, dfm4, dfm8], ignore_index=True)
dfm.shape
```

[4]:
```
(236, 3)
```

[8]:
```
# add epoch for each m
dfm['epoch'] = dfm.groupby(by=['m']).cumcount()
dfm.head()
```

[8]:

	m	val_accuracy	val_error	epoch
0	2	0.4884	0.5116	0
1	2	0.6893	0.3107	1
2	2	0.6408	0.3592	2
3	2	0.6936	0.3064	3
4	2	0.6094	0.3906	4

[9]:
```
dfm['epoch'] = dfm['epoch'] + 1
dfm.head()
```

[9]:

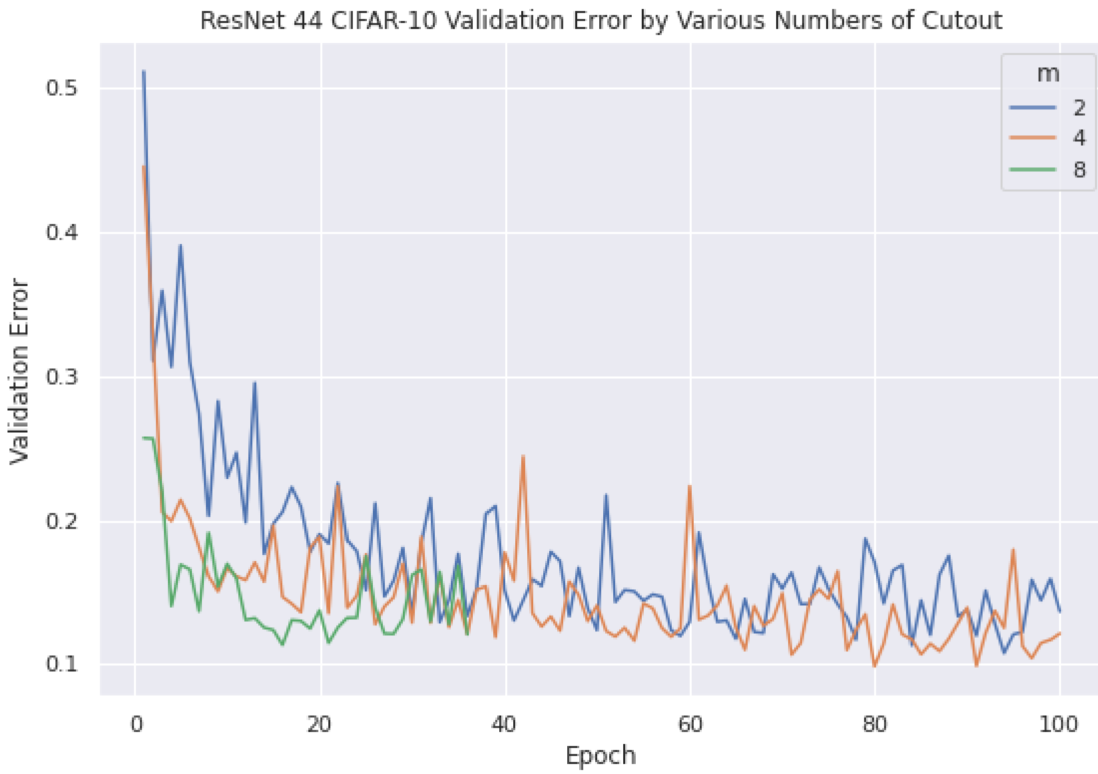
	m	val_accuracy	val_error	epoch
0	2	0.4884	0.5116	1
1	2	0.6893	0.3107	2
2	2	0.6408	0.3592	3
3	2	0.6936	0.3064	4
4	2	0.6094	0.3906	5

[14]:
```
dfm['m'] = dfm['m'].astype(str)
```

[15]:
```
import matplotlib.pyplot as plt
```

[17]:
```
fig, ax = plt.subplots(figsize=(9, 6))
sns.lineplot(x='epoch', y='val_error', hue='m', data=dfm)
ax.set_xlabel('Epoch')
ax.set_ylabel('Validation Error')
ax.set_title('ResNet 44 CIFAR-10 Validation Error by Various Numbers of Cutout')
```

[17]:
```
Text(0.5, 1.0, 'ResNet 44 CIFAR-10 Validation Error by Various Numbers of
Cutout')
```



The training time associated with various values of m are below:

m	training time (sec)	epochs	time/epoch (sec)
2	7496	100	~75
4	13861	100	~139
8	10291	36	~286

From the plot above, you can see that as m increases, the speed at which the model converges to the optimal accuracy decreases. Note: I trained each model with `EarlyStopping(patience=20)` and `m=8` stopped early.

1.1.5 Problem 5: *Universal Approximators: Depth vs Width (30 points)*

Q1 (20 points) Importing the necessary packages for this problem.

[ ]:
```
import numpy as np
from itertools import product
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
import tensorflow as tf
```

```
from tensorflow import keras
from tensorflow.keras import layers, Model
from time import time
from tqdm import tqdm

sns.set_theme(style="darkgrid")
```

Generate all possible combinations of network depths and unit combinations to evaluate on.

[ ]:
```
HIDDEN_UNITS = [2**n for n in range(4, 10)]
MAX_NUM_LAYERS = 3

UNITS = [list(c)
         for i in range(1, MAX_NUM_LAYERS + 1)
         for c in product(HIDDEN_UNITS, repeat=i)
         if sum(c) <= max(HIDDEN_UNITS)]

print(f"Number of unit combinations to test: {len(UNITS)}")
```

```
Number of unit combinations to test: 143
```

Define the `eggholder` and `y` functions.

[ ]:
```
def eggholder(x1, x2):
  left = - 1 * (x2 + 47) * np.sin(np.sqrt(np.abs((x1 / 2) + (x2 + 47))))
  right = x1 * np.sin(np.sqrt(np.abs(x1 - (x2 + 47))))
  return  left - right

def y(X):
  x1 = X[:, 0]
  x2 = X[:, 1]
  return eggholder(x1, x2) + np.random.normal(scale=0.3, size = x1.shape)
```

Randomly generate the input data `X`, pass through the objective function `y`, and perform the train-test split.

[ ]:
```
LOW = -512
HIGH = -1 * LOW
N = 100000
TEST_SPLIT = 0.2

X = np.random.randint(low=LOW, high=HIGH+1, size=(N, 2))
y = y(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =␣
 ↪TEST_SPLIT)
```

Create a function to dynamically build the model given a list of units.

```
[ ]: def build_model(units, model_enum):

       # build
       input = layers.Input(shape=(2, ), dtype='float32', name=f'input_{model_enum}')
       for i in range(len(units)):
         if i == 0:
           x = layers.Dense(units[i],
                            activation='relu',
                            name=f'dense_{model_enum}_{i}')(input)
         else:
           x = layers.BatchNormalization(name=f'norm_{model_enum}_{i}')(x)
           x = layers.Dense(units[i], activation='relu',␣
     ↪name=f'dense_{model_enum}_{i}')(x)
       output = layers.Dense(1, name=f'output_{model_enum}')(x)

       # compile
       model = Model(inputs=input, outputs=output, name=f'model_{model_enum}')
       opt = keras.optimizers.Nadam(name=f'nadam_{model_enum}')
       model.compile(loss='mse',
                     optimizer=opt,
                     metrics=[tf.keras.metrics.RootMeanSquaredError(f'rmse')])

       # return
       return model
```

Loop through each combination of units, train, and store metrics.

```
[ ]: DATA = []
     for i, units in enumerate(UNITS):
       print(f'{i+1}/{len(UNITS)}: {units} ...')
       model = build_model(units, i)
       t1 = time()
       history = model.fit(X_train, y_train,
                           batch_size=1000,
                           epochs=2000,
                           validation_data=(X_test, y_test),
                           verbose=0,
                           callbacks=[tf.keras.callbacks.
     ↪EarlyStopping(monitor='val_rmse',
                                                                            patience=20,
                                                                           ␣
     ↪restore_best_weights=True)])
       t2 = time()
       DATA.append({
           'units': units,
           'total_units': sum(units),
           'num_layers': len(units),
```

```
        'num_params': model.count_params(),
        'training_time': t2 - t1,
        'val_rmse': min(history.history.get('val_rmse'))
    })
```

```
1/143: [16] …
2/143: [32] …
3/143: [64] …
4/143: [128] …
5/143: [256] …
6/143: [512] …
7/143: [16, 16] …
8/143: [16, 32] …
9/143: [16, 64] …
10/143: [16, 128] …
11/143: [16, 256] …
12/143: [32, 16] …
13/143: [32, 32] …
14/143: [32, 64] …
15/143: [32, 128] …
16/143: [32, 256] …
17/143: [64, 16] …
18/143: [64, 32] …
19/143: [64, 64] …
20/143: [64, 128] …
21/143: [64, 256] …
22/143: [128, 16] …
23/143: [128, 32] …
24/143: [128, 64] …
25/143: [128, 128] …
26/143: [128, 256] …
27/143: [256, 16] …
28/143: [256, 32] …
29/143: [256, 64] …
30/143: [256, 128] …
31/143: [256, 256] …
32/143: [16, 16, 16] …
33/143: [16, 16, 32] …
34/143: [16, 16, 64] …
35/143: [16, 16, 128] …
36/143: [16, 16, 256] …
37/143: [16, 32, 16] …
38/143: [16, 32, 32] …
39/143: [16, 32, 64] …
40/143: [16, 32, 128] …
41/143: [16, 32, 256] …
42/143: [16, 64, 16] …
```

```
43/143: [16, 64, 32] …
44/143: [16, 64, 64] …
45/143: [16, 64, 128] …
46/143: [16, 64, 256] …
47/143: [16, 128, 16] …
48/143: [16, 128, 32] …
49/143: [16, 128, 64] …
50/143: [16, 128, 128] …
51/143: [16, 128, 256] …
52/143: [16, 256, 16] …
53/143: [16, 256, 32] …
54/143: [16, 256, 64] …
55/143: [16, 256, 128] …
56/143: [32, 16, 16] …
57/143: [32, 16, 32] …
58/143: [32, 16, 64] …
59/143: [32, 16, 128] …
60/143: [32, 16, 256] …
61/143: [32, 32, 16] …
62/143: [32, 32, 32] …
63/143: [32, 32, 64] …
64/143: [32, 32, 128] …
65/143: [32, 32, 256] …
66/143: [32, 64, 16] …
67/143: [32, 64, 32] …
68/143: [32, 64, 64] …
69/143: [32, 64, 128] …
70/143: [32, 64, 256] …
71/143: [32, 128, 16] …
72/143: [32, 128, 32] …
73/143: [32, 128, 64] …
74/143: [32, 128, 128] …
75/143: [32, 128, 256] …
76/143: [32, 256, 16] …
77/143: [32, 256, 32] …
78/143: [32, 256, 64] …
79/143: [32, 256, 128] …
80/143: [64, 16, 16] …
81/143: [64, 16, 32] …
82/143: [64, 16, 64] …
83/143: [64, 16, 128] …
84/143: [64, 16, 256] …
85/143: [64, 32, 16] …
86/143: [64, 32, 32] …
87/143: [64, 32, 64] …
88/143: [64, 32, 128] …
89/143: [64, 32, 256] …
90/143: [64, 64, 16] …
```

```
91/143: [64, 64, 32] …
92/143: [64, 64, 64] …
93/143: [64, 64, 128] …
94/143: [64, 64, 256] …
95/143: [64, 128, 16] …
96/143: [64, 128, 32] …
97/143: [64, 128, 64] …
98/143: [64, 128, 128] …
99/143: [64, 128, 256] …
100/143: [64, 256, 16] …
101/143: [64, 256, 32] …
102/143: [64, 256, 64] …
103/143: [64, 256, 128] …
104/143: [128, 16, 16] …
105/143: [128, 16, 32] …
106/143: [128, 16, 64] …
107/143: [128, 16, 128] …
108/143: [128, 16, 256] …
109/143: [128, 32, 16] …
110/143: [128, 32, 32] …
111/143: [128, 32, 64] …
112/143: [128, 32, 128] …
113/143: [128, 32, 256] …
114/143: [128, 64, 16] …
115/143: [128, 64, 32] …
116/143: [128, 64, 64] …
117/143: [128, 64, 128] …
118/143: [128, 64, 256] …
119/143: [128, 128, 16] …
120/143: [128, 128, 32] …
121/143: [128, 128, 64] …
122/143: [128, 128, 128] …
123/143: [128, 128, 256] …
124/143: [128, 256, 16] …
125/143: [128, 256, 32] …
126/143: [128, 256, 64] …
127/143: [128, 256, 128] …
128/143: [256, 16, 16] …
129/143: [256, 16, 32] …
130/143: [256, 16, 64] …
131/143: [256, 16, 128] …
132/143: [256, 32, 16] …
133/143: [256, 32, 32] …
134/143: [256, 32, 64] …
135/143: [256, 32, 128] …
136/143: [256, 64, 16] …
137/143: [256, 64, 32] …
138/143: [256, 64, 64] …
```

```
139/143: [256, 64, 128] …
140/143: [256, 128, 16] …
141/143: [256, 128, 32] …
142/143: [256, 128, 64] …
143/143: [256, 128, 128] …
```

Convert metrics to data frame and generate plots.

[ ]:
```python
df = pd.DataFrame(data=DATA)
df.head(20)
```

[ ]:

	units	total_units	num_layers	num_params	training_time	val_rmse
0	[16]	16	1	65	103.268659	294.535004
1	[32]	32	1	129	167.009094	286.857635
2	[64]	64	1	257	91.885740	288.383728
3	[128]	128	1	513	201.973639	280.637390
4	[256]	256	1	1025	97.096044	283.629059
5	[512]	512	1	2049	49.887353	287.169952
6	[16, 16]	32	2	401	133.961195	196.519699
7	[16, 32]	48	2	689	185.208984	162.904724
8	[16, 64]	80	2	1265	93.145822	179.190598
9	[16, 128]	144	2	2417	215.616152	149.791107
10	[16, 256]	272	2	4721	138.178831	140.212112
11	[32, 16]	48	2	769	76.988253	213.370163
12	[32, 32]	64	2	1313	104.727805	183.775284
13	[32, 64]	96	2	2401	82.812294	162.609573
14	[32, 128]	160	2	4577	132.551145	130.526886
15	[32, 256]	288	2	8929	201.226646	111.449539
16	[64, 16]	80	2	1505	80.537540	190.546463
17	[64, 32]	96	2	2561	68.448728	161.412048
18	[64, 64]	128	2	4673	90.081338	146.507324
19	[64, 128]	192	2	8897	88.286884	146.440613

[ ]:
```python
# aggregate by number of units and depth
df_g = df.groupby(['total_units', 'num_layers'])['val_rmse'].mean()
df_g = df_g.reset_index()
df_g.head(20)
```

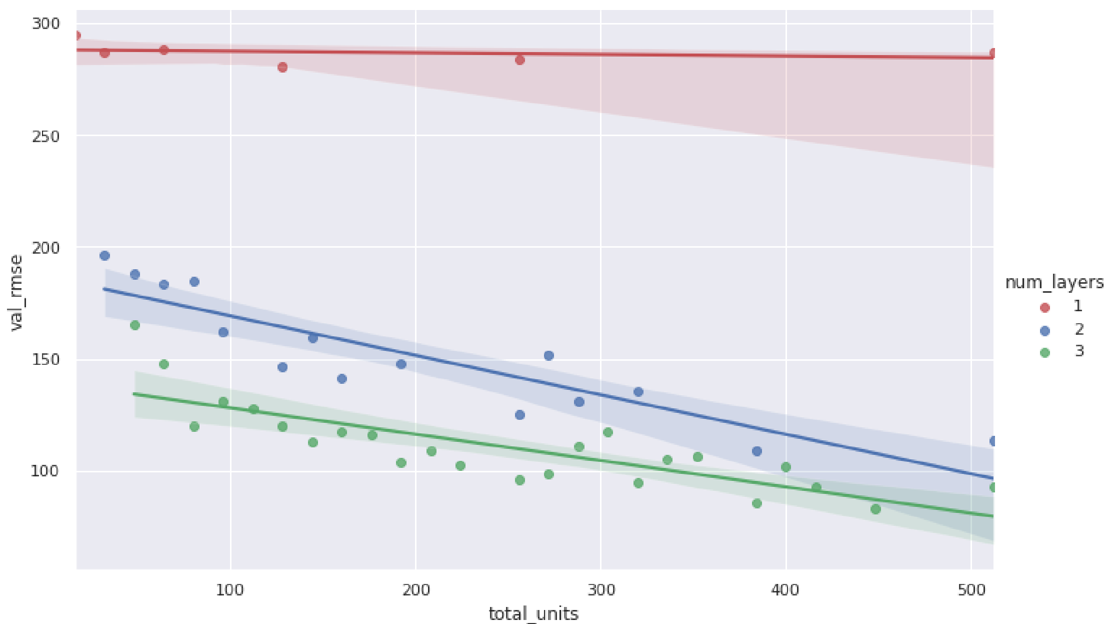
[ ]:

	total_units	num_layers	val_rmse
0	16	1	294.535004
1	32	1	286.857635
2	32	2	196.519699
3	48	2	188.137444
4	48	3	165.131546
5	64	1	288.383728
6	64	2	183.775284
7	64	3	147.934794
8	80	2	184.868530

```
9            80          3   120.197194
10           96          2   162.010811
11           96          3   130.856380
12          112          3   127.977982
13          128          1   280.637390
14          128          2   146.507324
15          128          3   120.213455
16          144          2   159.455887
17          144          3   113.233665
18          160          2   141.679184
19          160          3   117.377724
```

[ ]:
```python
# plot validation error metric vs units by depth
sns.lmplot(x='total_units',
           y='val_rmse',
           hue='num_layers',
           data=df_g,
           legend='full',
           palette=['r', 'b', 'g'],
           height=6,
           aspect=1.6)
```

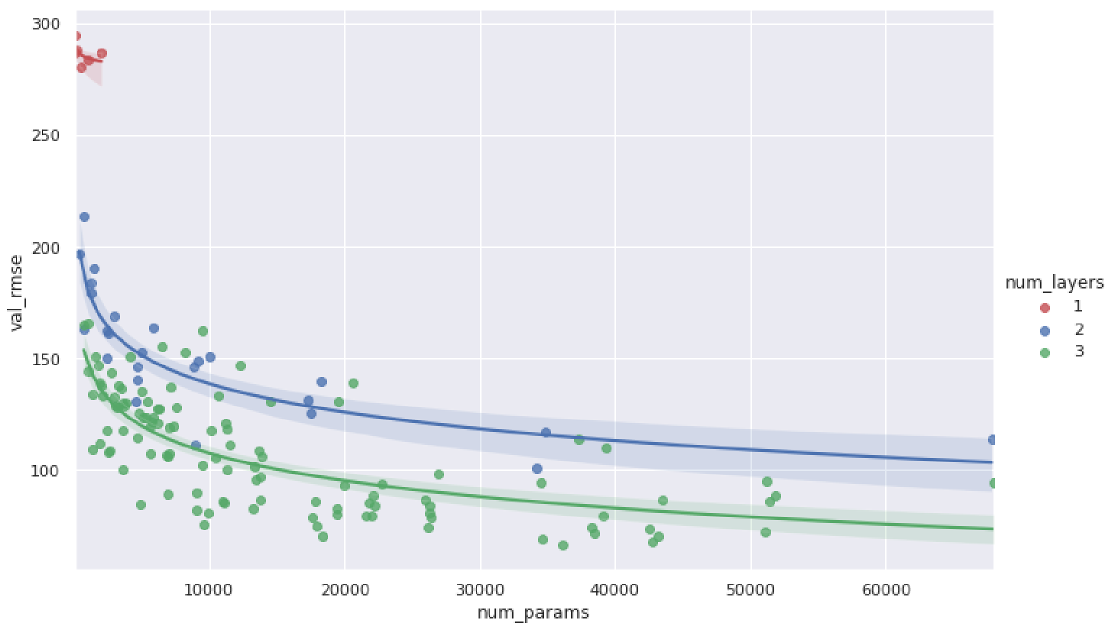
[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7f6f5810ef98>
```



[ ]:
```python
# plot validation error metric vs parameters by depth
sns.lmplot(x='num_params',
```

```
        y='val_rmse',
        hue='num_layers',
        data=df,
        legend='full',
        palette=['r', 'b', 'g'],
        height=6,
        aspect=1.6,
        logx=True)
```

[ ]: <seaborn.axisgrid.FacetGrid at 0x7f6f57f62da0>



Q2: (10 points) From the two plots above, we can make the following observations:

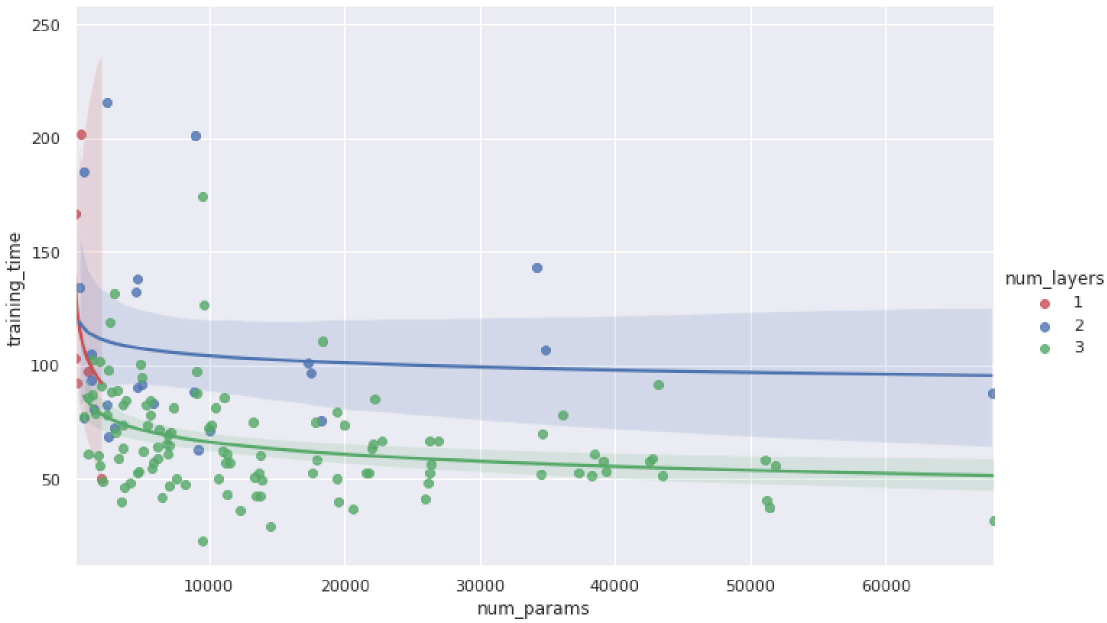
1. In general, the validation RMSE decreases as more units are introduced into the network. The property holds for networks of hidden layer size 1, 2, and 3.
2. In general, the validation RMSE decreases as more parameters are introduced into the network. The property holds for networks of hidden layer size 1, 2, and 3. There is a leveling off in propensity to learn as more parameters are added however, for all of the different layer sized networks, which can be seen in the second figure from Q1. Number of parameters is very similar to the number of units in this sense.
3. On the hole, deeper networks possess the ability to achieve better results, as validation RMSE drops in a stepwise manner from networks of depths 1, 2, and 3.

A plot of training time versus the number of parameters is below.

[ ]:

```
# plot training time vs parameters by depth
sns.lmplot(x='num_params',
           y='training_time',
           hue='num_layers',
           data=df,
           legend='full',
           palette=['r', 'b', 'g'],
           height=6,
           aspect=1.6, logx=True)
```

[ ]: <seaborn.axisgrid.FacetGrid at 0x7f6f5a55a2e8>



From above we can see that there is a s similar relationship in training time versus number of layers as there was for validation RMSE and number of layers.

Intuitively, this makes sense. Even though the deeper networks have larger number of parameters, they can learn the objective function and converge much faster than more shallow networks.