dlnd_image_classification

March 28, 2017

1 Image Classification

In this project, you'll classify images from the CIFAR-10 dataset. The dataset consists of airplanes, dogs, cats, and other objects. You'll preprocess the images, then train a convolutional neural network on all the samples. The images need to be normalized and the labels need to be one-hot encoded. You'll get to apply what you learned and build a convolutional, max pooling, dropout, and fully connected layers. At the end, you'll get to see your neural network's predictions on the sample images. ## Get the Data Run the following cell to download the CIFAR-10 dataset for python.

In [1]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
from urllib.request import urlretrieve
from os.path import isfile, isdir
from tqdm import tqdm
import problem_unittests as tests
import tarfile

cifar10_dataset_folder_path = 'cifar-10-batches-py'

class DLProgress(tqdm):
    last_block = 0

    def hook(self, block_num=1, block_size=1, total_size=None):
        self.total = total_size
        self.update((block_num - self.last_block) * block_size)
        self.last_block = block_num

if not isfile('cifar-10-python.tar.gz'):
    with DLProgress(unit='B', unit_scale=True, miniters=1, desc='CIFAR-10 Dataset') as p
        urlretrieve(
            'https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz',
            'cifar-10-python.tar.gz',
            pbar.hook)

if not isdir(cifar10_dataset_folder_path):
```

```
            with tarfile.open('cifar-10-python.tar.gz') as tar:
                tar.extractall()
                tar.close()


        tests.test_folder_path(cifar10_dataset_folder_path)
```

```
CIFAR-10 Dataset: 171MB [00:52, 3.89MB/s]


All files found!
```

1.1 Explore the Data

The dataset is broken into batches to prevent your machine from running out of memory. The CIFAR-10 dataset consists of 5 batches, named `data_batch_1`, `data_batch_2`, etc.. Each batch contains the labels and images that are one of the following: * airplane * automobile * bird * cat * deer * dog * frog * horse * ship * truck

Understanding a dataset is part of making predictions on the data. Play around with the code cell below by changing the `batch_id` and `sample_id`. The `batch_id` is the id for a batch (1-5). The `sample_id` is the id for a image and label pair in the batch.

Ask yourself "What are all possible labels?", "What is the range of values for the image data?", "Are the labels in order or random?". Answers to questions like these will help you preprocess the data and end up with better predictions.

```
In [5]: %matplotlib inline
        %config InlineBackend.figure_format = 'retina'

        import helper
        import numpy as np

        # Explore the dataset
        batch_id = 3
        sample_id = 1000
        helper.display_stats(cifar10_dataset_folder_path, batch_id, sample_id)
```

```
Stats of batch 3:
Samples: 10000
Label Counts: {0: 994, 1: 1042, 2: 965, 3: 997, 4: 990, 5: 1029, 6: 978, 7: 1015, 8: 961, 9: 102
First 20 Labels: [8, 5, 0, 6, 9, 2, 8, 3, 6, 2, 7, 4, 6, 9, 0, 0, 7, 3, 7, 2]

Example of Image 1000:
Image - Min Value: 37 Max Value: 230
Image - Shape: (32, 32, 3)
Label - Label Id: 0 Name: airplane
```

1.2 Implement Preprocess Functions

1.2.1 Normalize

In the cell below, implement the `normalize` function to take in image data, `x`, and return it as a normalized Numpy array. The values should be in the range of 0 to 1, inclusive. The return object should be the same shape as `x`.

```
In [10]: def normalize(x):
             """
             Normalize a list of sample image data in the range of 0 to 1
             : x: List of image data.  The image shape is (32, 32, 3)
             : return: Numpy array of normalize data
             """
             # TODO: Implement Function
             return x/255.0


         """
         DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
         """
         tests.test_normalize(normalize)
```

```
Tests Passed
```

1.2.2 One-hot encode

Just like the previous code cell, you'll be implementing a function for preprocessing. This time, you'll implement the `one_hot_encode` function. The input, `x`, are a list of labels. Implement the function to return the list of labels as One-Hot encoded Numpy array. The possible values for labels are 0 to 9. The one-hot encoding function should return the same encoding for each value between each call to `one_hot_encode`. Make sure to save the map of encodings outside the function.

Hint: Don't reinvent the wheel.

```
In [29]: def one_hot_encode(x):
             """
             One hot encode a list of sample labels. Return a one-hot encoded vector for each la
             : x: List of sample Labels
             : return: Numpy array of one-hot encoded labels
             """
             # TODO: Implement Function
         #     one_hot = np.zeros((len(x), 10))
         #     for i, j in enumerate(x):
         #         one_hot[i][j]=1
             one_hot = np.eye(10)[x]
             return one_hot


         """
         DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
         """
         tests.test_one_hot_encode(one_hot_encode)
```

```
Tests Passed
```

1.2.3 Randomize Data

As you saw from exploring the data above, the order of the samples are randomized. It doesn't hurt to randomize it again, but you don't need to for this dataset.

1.3 Preprocess all the data and save it

Running the code cell below will preprocess all the CIFAR-10 data and save it to file. The code below also uses 10% of the training data for validation.

```
In [30]: """
         DON'T MODIFY ANYTHING IN THIS CELL
         """
         # Preprocess Training, Validation, and Testing Data
         helper.preprocess_and_save_data(cifar10_dataset_folder_path, normalize, one_hot_encode)
```

2 Check Point

This is your first checkpoint. If you ever decide to come back to this notebook or have to restart the notebook, you can start from here. The preprocessed data has been saved to disk.

```
In [102]: """
          DON'T MODIFY ANYTHING IN THIS CELL
          """
          import pickle
          import problem_unittests as tests
          import helper
          import functools
          # Load the Preprocessed Validation data
          valid_features, valid_labels = pickle.load(open('preprocess_validation.p', mode='rb'))
```

2.1 Build the network

For the neural network, you'll build each layer into a function. Most of the code you've seen has been outside of functions. To test your code more thoroughly, we require that you put each layer in a function. This allows us to give you better feedback and test for simple mistakes using our unittests before you submit your project.

> **Note:** If you're finding it hard to dedicate enough time for this course each week, we've provided a small shortcut to this part of the project. In the next couple of problems, you'll have the option to use classes from the TensorFlow Layers or TensorFlow Layers (contrib) packages to build each layer, except the layers you build in the "Convolutional and Max Pooling Layer" section. TF Layers is similar to Keras's and TFLearn's abstraction to layers, so it's easy to pickup.
>
> However, if you would like to get the most out of this course, try to solve all the problems *without* using anything from the TF Layers packages. You **can** still use classes from other packages that happen to have the same name as ones you find in TF Layers! For example, instead of using the TF Layers version of the `conv2d` class, tf.layers.conv2d, you would want to use the TF Neural Network version of `conv2d`, tf.nn.conv2d.

Let's begin!

2.1.1 Input

The neural network needs to read the image data, one-hot encoded labels, and dropout keep probability. Implement the following functions * Implement `neural_net_image_input` * Return a TF Placeholder * Set the shape using `image_shape` with batch size set to `None`. * Name the TensorFlow placeholder "x" using the TensorFlow `name` parameter in the TF Placeholder. * Implement `neural_net_label_input` * Return a TF Placeholder * Set the shape using `n_classes` with batch size set to `None`. * Name the TensorFlow placeholder "y" using the TensorFlow `name` parameter in the TF Placeholder. * Implement `neural_net_keep_prob_input` * Return a TF Placeholder for dropout keep probability. * Name the TensorFlow placeholder "keep_prob" using the TensorFlow `name` parameter in the TF Placeholder.

These names will be used at the end of the project to load your saved model.
Note: `None` for shapes in TensorFlow allow for a dynamic size.

In [40]:
```
import tensorflow as tf

def neural_net_image_input(image_shape):
    """
    Return a Tensor for a batch of image input
    : image_shape: Shape of the images
    : return: Tensor for image input.
    """
    # TODO: Implement Function
    return tf.placeholder(tf.float32, (None,)+image_shape, "x")


def neural_net_label_input(n_classes):
    """
    Return a Tensor for a batch of label input
    : n_classes: Number of classes
    : return: Tensor for label input.
    """
    # TODO: Implement Function
    return tf.placeholder(tf.float32, (None, n_classes), "y")


def neural_net_keep_prob_input():
    """
    Return a Tensor for keep probability
    : return: Tensor for keep probability.
    """
    # TODO: Implement Function
    return tf.placeholder(tf.float32, name="keep_prob")


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tf.reset_default_graph()
tests.test_nn_image_inputs(neural_net_image_input)
tests.test_nn_label_inputs(neural_net_label_input)
tests.test_nn_keep_prob_inputs(neural_net_keep_prob_input)
```

```
Image Input Tests Passed.
Label Input Tests Passed.
Keep Prob Tests Passed.
```

2.1.2 Convolution and Max Pooling Layer

Convolution layers have a lot of success with images. For this code cell, you should implement the function `conv2d_maxpool` to apply convolution then max pooling: * Create the weight and bias using `conv_ksize`, `conv_num_outputs` and the shape of `x_tensor`. * Apply a convolution to `x_tensor` using weight and `conv_strides`. * We recommend you use same padding, but you're welcome to use any padding. * Add bias * Add a nonlinear activation to the convolution. * Apply Max Pooling using `pool_ksize` and `pool_strides`. * We recommend you use same padding, but you're welcome to use any padding.

Note: You **can't** use TensorFlow Layers or TensorFlow Layers (contrib) for **this** layer, but you can still use TensorFlow's Neural Network package. You may still use the shortcut option for all the **other** layers.

```
In [180]: def conv2d_maxpool(x_tensor, conv_num_outputs, conv_ksize, conv_strides, pool_ksize, p
              """
              Apply convolution then max pooling to x_tensor
              :param x_tensor: TensorFlow Tensor
              :param conv_num_outputs: Number of outputs for the convolutional layer
              :param conv_ksize: kernal size 2-D Tuple for the convolutional layer
              :param conv_strides: Stride 2-D Tuple for convolution
              :param pool_ksize: kernal size 2-D Tuple for pool
              :param pool_strides: Stride 2-D Tuple for pool
              : return: A tensor that represents convolution and max pooling of x_tensor
              """
              # TODO: Implement Function
              weights_shape = conv_ksize + (x_tensor.shape[3].value, conv_num_outputs)
              weights = tf.Variable(tf.truncated_normal(weights_shape, stddev=0.1), name="conv_w
              bias = tf.Variable(tf.zeros(conv_num_outputs))

              conv_strides = (1,) + conv_strides + (1,)
              pool_strides = (1,) + pool_strides + (1,)
              pool_size = (1,) + pool_ksize + (1,)

              conv_layer = tf.nn.conv2d(x_tensor, weights, conv_strides, "SAME")
              conv_layer = tf.nn.bias_add(conv_layer, bias)
              conv_layer = tf.nn.relu(conv_layer)
              conv_layer = tf.nn.max_pool(conv_layer, pool_size, pool_strides, "SAME", name="con
              return conv_layer


          """
          DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
          """
          tests.test_con_pool(conv2d_maxpool)

Tests Passed
```

2.1.3 Flatten Layer

Implement the `flatten` function to change the dimension of `x_tensor` from a 4-D tensor to a 2-D tensor. The output should be the shape (*Batch Size*, *Flattened Image Size*). Shortcut option: you can use classes from the TensorFlow Layers or TensorFlow Layers (contrib) packages for this layer. For more of a challenge, only use other TensorFlow packages.

```
In [181]: def flatten(x_tensor):
              """
              Flatten x_tensor to (Batch Size, Flattened Image Size)
              : x_tensor: A tensor of size (Batch Size, ...), where ... are the image dimensions
              : return: A tensor of size (Batch Size, Flattened Image Size).
              """
              # TODO: Implement Function

              x = tf.convert_to_tensor(x_tensor)
              batch_dim = tf.slice(tf.shape(x), [0], [1])
              x_dim = tf.slice(tf.shape(x), [1], [x.shape.ndims-1])

              flatten_dim = tf.expand_dims(tf.reduce_prod(x_dim), 0)
              flatten_shape = tf.concat([batch_dim, flatten_dim], 0)

              flatten = tf.reshape(x, flatten_shape, name="flatten_layer")

              x_shape = x.shape.as_list()
              batch_dim, spatial_dims = x_shape[0], x_shape[1:]
              if all(spatial_dims):
                  flatten.set_shape([batch_dim,
                                     functools.reduce(lambda x, y: x * y, spatial_dims)])
              else:
                  flatten.set_shape([batch_dim, None])

          #   flatten = tf.contrib.layers.flatten(x_tensor)
              return flatten


          """
          DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
          """
          tests.test_flatten(flatten)
```

```
Tests Passed
```

2.1.4 Fully-Connected Layer

Implement the `fully_conn` function to apply a fully connected layer to `x_tensor` with the shape (*Batch Size*, *num_outputs*). Shortcut option: you can use classes from the TensorFlow Layers or

TensorFlow Layers (contrib) packages for this layer. For more of a challenge, only use other TensorFlow packages.

```
In [182]: def fully_conn(x_tensor, num_outputs):
              """
              Apply a fully connected layer to x_tensor using weight and bias
              : x_tensor: A 2-D tensor where the first dimension is batch size.
              : num_outputs: The number of output that the new tensor should be.
              : return: A 2-D tensor where the second dimension is num_outputs.
              """
              # TODO: Implement Function
              weights_shape = (x_tensor.shape[1].value, num_outputs)
              weights = tf.Variable(tf.truncated_normal(weights_shape, stddev=0.1), name="fully_
              bias = tf.Variable(tf.zeros(num_outputs), name="fully_conn_bias")

              fully_conn_layer = tf.matmul(x_tensor, weights)
              fully_conn_layer = tf.nn.bias_add(fully_conn_layer, bias, name="fully_conn_layer")
              fully_conn_layer = tf.nn.relu(fully_conn_layer, name="fully_conn_layer")
              return fully_conn_layer

          """
          DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
          """
          tests.test_fully_conn(fully_conn)
```

```
Tests Passed
```

2.1.5 Output Layer

Implement the `output` function to apply a fully connected layer to `x_tensor` with the shape (*Batch Size*, *num_outputs*). Shortcut option: you can use classes from the TensorFlow Layers or TensorFlow Layers (contrib) packages for this layer. For more of a challenge, only use other TensorFlow packages.

Note: Activation, softmax, or cross entropy should **not** be applied to this.

```
In [185]: def output(x_tensor, num_outputs):
              """
              Apply a output layer to x_tensor using weight and bias
              : x_tensor: A 2-D tensor where the first dimension is batch size.
              : num_outputs: The number of output that the new tensor should be.
              : return: A 2-D tensor where the second dimension is num_outputs.
              """
              # TODO: Implement Function
              weights_shape = (x_tensor.shape[1].value, num_outputs)
              weights = tf.Variable(tf.truncated_normal(weights_shape, stddev=0.1), name="output
              bias = tf.Variable(tf.zeros(num_outputs), name="output_bias")

              output_layer = tf.matmul(x_tensor, weights)
```

```
    output_layer = tf.nn.bias_add(output_layer, bias, name="output_layer")
    return output_layer

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_output(output)
```

```
Tests Passed
```

2.1.6 Create Convolutional Model

Implement the function `conv_net` to create a convolutional neural network model. The function takes in a batch of images, `x`, and outputs logits. Use the layers you created above to create this model:

- Apply 1, 2, or 3 Convolution and Max Pool layers
- Apply a Flatten Layer
- Apply 1, 2, or 3 Fully Connected Layers
- Apply an Output Layer
- Return the output
- Apply TensorFlow's Dropout to one or more layers in the model using `keep_prob`.

```
In [186]: def conv_net(x, keep_prob):
              """
              Create a convolutional neural network model
              : x: Placeholder tensor that holds image data.
              : keep_prob: Placeholder tensor that hold dropout keep probability.
              : return: Tensor that represents logits
              """
              # TODO: Apply 1, 2, or 3 Convolution and Max Pool layers
              #    Play around with different number of outputs, kernel size and stride
              # Function Definition from Above:
              #    conv2d_maxpool(x_tensor, conv_num_outputs, conv_ksize, conv_strides, pool_ksi
              conv_num_outputs_list = [16, 32, 64]
              conv_ksize = (4, 4)
              conv_strides  = (1, 1)
              pool_ksize = (2, 2)
              pool_strides  = (2, 2)

              conv_layer_1 = conv2d_maxpool(x, conv_num_outputs_list[0], conv_ksize, conv_stride
              conv_layer_2 = conv2d_maxpool(conv_layer_1, conv_num_outputs_list[1], conv_ksize,
              conv_layer_3 = conv2d_maxpool(conv_layer_2, conv_num_outputs_list[2], conv_ksize,


              # TODO: Apply a Flatten Layer
              # Function Definition from Above:
              #   flatten(x_tensor)
```

```
    flatten_layer = flatten(conv_layer_3)

    # TODO: Apply 1, 2, or 3 Fully Connected Layers
    #    Play around with different number of outputs
    # Function Definition from Above:
    #   fully_conn(x_tensor, num_outputs)
    fully_conn_layer = fully_conn(flatten_layer, 10)
    fully_conn_layer = tf.nn.dropout(fully_conn_layer, keep_prob)
    # TODO: Apply an Output Layer
    #    Set this to the number of classes
    # Function Definition from Above:
    #   output(x_tensor, num_outputs)
    output_layer = output(fully_conn_layer, 10)

    # TODO: return output
    return output_layer


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""

##############################
## Build the Neural Network ##
##############################

# Remove previous weights, bias, inputs, etc..
tf.reset_default_graph()

# Inputs
x = neural_net_image_input((32, 32, 3))
y = neural_net_label_input(10)
keep_prob = neural_net_keep_prob_input()

# Model
logits = conv_net(x, keep_prob)

# Name logits Tensor, so that is can be loaded from disk after training
logits = tf.identity(logits, name='logits')

# Loss and Optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=y))
optimizer = tf.train.AdamOptimizer().minimize(cost)

# Accuracy
correct_pred = tf.equal(tf.argmax(logits, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32), name='accuracy')
```

```
tests.test_conv_net(conv_net)
```

```
Neural Network Built!
```

2.2 Train the Neural Network

2.2.1 Single Optimization

Implement the function `train_neural_network` to do a single optimization. The optimization should use `optimizer` to optimize in `session` with a `feed_dict` of the following: * `x` for image input * `y` for labels * `keep_prob` for keep probability for dropout

This function will be called for each batch, so `tf.global_variables_initializer()` has already been called.

Note: Nothing needs to be returned. This function is only optimizing the neural network.

```
In [187]: def train_neural_network(session, optimizer, keep_probability, feature_batch, label_ba
              """
              Optimize the session on a batch of images and labels
              : session: Current TensorFlow session
              : optimizer: TensorFlow optimizer function
              : keep_probability: keep probability
              : feature_batch: Batch of Numpy image data
              : label_batch: Batch of Numpy label data
              """
              # TODO: Implement Function
              session.run(optimizer, feed_dict={x: feature_batch, y: label_batch, keep_prob: kee


          """
          DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
          """
          tests.test_train_nn(train_neural_network)
```

```
Tests Passed
```

2.2.2 Show Stats

Implement the function `print_stats` to print loss and validation accuracy. Use the global variables `valid_features` and `valid_labels` to calculate validation accuracy. Use a keep probability of `1.0` to calculate the loss and validation accuracy.

```
In [188]: def print_stats(session, feature_batch, label_batch, cost, accuracy):
              """
              Print information about loss and validation accuracy
              : session: Current TensorFlow session
              : feature_batch: Batch of Numpy image data
              : label_batch: Batch of Numpy label data
```

```
    : cost: TensorFlow cost function
    : accuracy: TensorFlow accuracy function
    """
    # TODO: Implement Function
    loss = session.run(cost, feed_dict={x: feature_batch, y:label_batch, keep_prob: 1.
    valid_accuracy = session.run(accuracy, feed_dict={x: valid_features, y:valid_label
    print("Loss: {:>6.4f}  Validation Accuracy: {:.4f}".format(loss, valid_accuracy))
```

2.2.3 Hyperparameters

Tune the following parameters: * Set `epochs` to the number of iterations until the network stops learning or start overfitting * Set `batch_size` to the highest number that your machine has memory for. Most people set them to common sizes of memory: * 64 * 128 * 256 * ... * Set `keep_probability` to the probability of keeping a node using dropout

In [191]:
```
# TODO: Tune Parameters
epochs = 100
batch_size = 256
keep_probability = 0.8
```

2.2.4 Train on a Single CIFAR-10 Batch

Instead of training the neural network on all the CIFAR-10 batches of data, let's use a single batch. This should save time while you iterate on the model to get a better accuracy. Once the final validation accuracy is 50% or greater, run the model on all the data in the next section.

In [170]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
print('Checking the Training on a Single Batch...')
with tf.Session() as sess:
    # Initializing the variables
    sess.run(tf.global_variables_initializer())

    # Training cycle
    for epoch in range(epochs):
        batch_i = 1
        for batch_features, batch_labels in helper.load_preprocess_training_batch(batc
            train_neural_network(sess, optimizer, keep_probability, batch_features, ba
        print('Epoch {:>2}, CIFAR-10 Batch {}:  '.format(epoch + 1, batch_i), end='')
        print_stats(sess, batch_features, batch_labels, cost, accuracy)
```

```
Checking the Training on a Single Batch...
Epoch  1, CIFAR-10 Batch 1:  Loss:     2.3026     Validation Accuracy: 0.098800
Epoch  2, CIFAR-10 Batch 1:  Loss:     2.3026     Validation Accuracy: 0.099000
Epoch  3, CIFAR-10 Batch 1:  Loss:     2.3026     Validation Accuracy: 0.099000
Epoch  4, CIFAR-10 Batch 1:  Loss:     2.3026     Validation Accuracy: 0.098400
Epoch  5, CIFAR-10 Batch 1:  Loss:     2.3026     Validation Accuracy: 0.098400
Epoch  6, CIFAR-10 Batch 1:  Loss:     2.3026     Validation Accuracy: 0.098400
```

```
Epoch  7, CIFAR-10 Batch 1:  Loss:     2.3026     Validation Accuracy: 0.098400
Epoch  8, CIFAR-10 Batch 1:  Loss:     2.3026     Validation Accuracy: 0.098400
Epoch  9, CIFAR-10 Batch 1:  Loss:     2.3026     Validation Accuracy: 0.098400
Epoch 10, CIFAR-10 Batch 1:  Loss:     2.3026     Validation Accuracy: 0.098200
```

2.2.5 Fully Train the Model

Now that you got a good accuracy with a single CIFAR-10 batch, try it with all five batches.

```
In [192]: """
          DON'T MODIFY ANYTHING IN THIS CELL
          """
          save_model_path = './image_classification'

          print('Training...')
          with tf.Session() as sess:
              # Initializing the variables
              sess.run(tf.global_variables_initializer())

              # Training cycle
              for epoch in range(epochs):
                  # Loop over all batches
                  n_batches = 5
                  for batch_i in range(1, n_batches + 1):
                      for batch_features, batch_labels in helper.load_preprocess_training_batch(
                          train_neural_network(sess, optimizer, keep_probability, batch_features
                      print('Epoch {:>2}, CIFAR-10 Batch {}:  '.format(epoch + 1, batch_i), end=
                      print_stats(sess, batch_features, batch_labels, cost, accuracy)

              # Save Model
              saver = tf.train.Saver()
              save_path = saver.save(sess, save_model_path)
```

```
Training...
Epoch  1, CIFAR-10 Batch 1:  Loss: 2.2472  Validation Accuracy: 0.1652
Epoch  1, CIFAR-10 Batch 2:  Loss: 2.2197  Validation Accuracy: 0.2520
Epoch  1, CIFAR-10 Batch 3:  Loss: 1.8955  Validation Accuracy: 0.2932
Epoch  1, CIFAR-10 Batch 4:  Loss: 1.8806  Validation Accuracy: 0.3226
Epoch  1, CIFAR-10 Batch 5:  Loss: 1.8179  Validation Accuracy: 0.3596
Epoch  2, CIFAR-10 Batch 1:  Loss: 1.9865  Validation Accuracy: 0.3766
Epoch  2, CIFAR-10 Batch 2:  Loss: 1.7393  Validation Accuracy: 0.4066
Epoch  2, CIFAR-10 Batch 3:  Loss: 1.4919  Validation Accuracy: 0.4078
Epoch  2, CIFAR-10 Batch 4:  Loss: 1.5969  Validation Accuracy: 0.4258
Epoch  2, CIFAR-10 Batch 5:  Loss: 1.6071  Validation Accuracy: 0.4172
Epoch  3, CIFAR-10 Batch 1:  Loss: 1.7288  Validation Accuracy: 0.4488
Epoch  3, CIFAR-10 Batch 2:  Loss: 1.5289  Validation Accuracy: 0.4710
Epoch  3, CIFAR-10 Batch 3:  Loss: 1.2569  Validation Accuracy: 0.4420
```

```
Epoch  3, CIFAR-10 Batch 4:  Loss: 1.4595  Validation Accuracy: 0.4742
Epoch  3, CIFAR-10 Batch 5:  Loss: 1.4661  Validation Accuracy: 0.4684
Epoch  4, CIFAR-10 Batch 1:  Loss: 1.6106  Validation Accuracy: 0.4840
Epoch  4, CIFAR-10 Batch 2:  Loss: 1.4443  Validation Accuracy: 0.4788
Epoch  4, CIFAR-10 Batch 3:  Loss: 1.1750  Validation Accuracy: 0.4674
Epoch  4, CIFAR-10 Batch 4:  Loss: 1.3708  Validation Accuracy: 0.4972
Epoch  4, CIFAR-10 Batch 5:  Loss: 1.3299  Validation Accuracy: 0.4906
Epoch  5, CIFAR-10 Batch 1:  Loss: 1.5095  Validation Accuracy: 0.5022
Epoch  5, CIFAR-10 Batch 2:  Loss: 1.3289  Validation Accuracy: 0.5068
Epoch  5, CIFAR-10 Batch 3:  Loss: 1.1163  Validation Accuracy: 0.4966
Epoch  5, CIFAR-10 Batch 4:  Loss: 1.3153  Validation Accuracy: 0.5136
Epoch  5, CIFAR-10 Batch 5:  Loss: 1.2742  Validation Accuracy: 0.5222
Epoch  6, CIFAR-10 Batch 1:  Loss: 1.4295  Validation Accuracy: 0.4916
Epoch  6, CIFAR-10 Batch 2:  Loss: 1.2460  Validation Accuracy: 0.5116
Epoch  6, CIFAR-10 Batch 3:  Loss: 1.0543  Validation Accuracy: 0.5184
Epoch  6, CIFAR-10 Batch 4:  Loss: 1.2798  Validation Accuracy: 0.5194
Epoch  6, CIFAR-10 Batch 5:  Loss: 1.2061  Validation Accuracy: 0.5236
Epoch  7, CIFAR-10 Batch 1:  Loss: 1.3290  Validation Accuracy: 0.5198
Epoch  7, CIFAR-10 Batch 2:  Loss: 1.1793  Validation Accuracy: 0.5230
Epoch  7, CIFAR-10 Batch 3:  Loss: 1.0385  Validation Accuracy: 0.5258
Epoch  7, CIFAR-10 Batch 4:  Loss: 1.2688  Validation Accuracy: 0.5262
Epoch  7, CIFAR-10 Batch 5:  Loss: 1.1472  Validation Accuracy: 0.5436
Epoch  8, CIFAR-10 Batch 1:  Loss: 1.2780  Validation Accuracy: 0.5190
Epoch  8, CIFAR-10 Batch 2:  Loss: 1.0998  Validation Accuracy: 0.5406
Epoch  8, CIFAR-10 Batch 3:  Loss: 1.0267  Validation Accuracy: 0.5346
Epoch  8, CIFAR-10 Batch 4:  Loss: 1.2033  Validation Accuracy: 0.5396
Epoch  8, CIFAR-10 Batch 5:  Loss: 1.0755  Validation Accuracy: 0.5492
Epoch  9, CIFAR-10 Batch 1:  Loss: 1.2058  Validation Accuracy: 0.5460
Epoch  9, CIFAR-10 Batch 2:  Loss: 1.0946  Validation Accuracy: 0.5558
Epoch  9, CIFAR-10 Batch 3:  Loss: 0.9586  Validation Accuracy: 0.5426
Epoch  9, CIFAR-10 Batch 4:  Loss: 1.1376  Validation Accuracy: 0.5576
Epoch  9, CIFAR-10 Batch 5:  Loss: 1.0260  Validation Accuracy: 0.5566
Epoch 10, CIFAR-10 Batch 1:  Loss: 1.1731  Validation Accuracy: 0.5382
Epoch 10, CIFAR-10 Batch 2:  Loss: 1.0245  Validation Accuracy: 0.5684
Epoch 10, CIFAR-10 Batch 3:  Loss: 0.9013  Validation Accuracy: 0.5590
Epoch 10, CIFAR-10 Batch 4:  Loss: 1.1070  Validation Accuracy: 0.5620
Epoch 10, CIFAR-10 Batch 5:  Loss: 0.9483  Validation Accuracy: 0.5662
Epoch 11, CIFAR-10 Batch 1:  Loss: 1.0874  Validation Accuracy: 0.5576
Epoch 11, CIFAR-10 Batch 2:  Loss: 0.9886  Validation Accuracy: 0.5732
Epoch 11, CIFAR-10 Batch 3:  Loss: 0.9538  Validation Accuracy: 0.5568
Epoch 11, CIFAR-10 Batch 4:  Loss: 1.0513  Validation Accuracy: 0.5768
Epoch 11, CIFAR-10 Batch 5:  Loss: 0.9476  Validation Accuracy: 0.5678
Epoch 12, CIFAR-10 Batch 1:  Loss: 1.0451  Validation Accuracy: 0.5598
Epoch 12, CIFAR-10 Batch 2:  Loss: 0.9277  Validation Accuracy: 0.5818
Epoch 12, CIFAR-10 Batch 3:  Loss: 0.8347  Validation Accuracy: 0.5772
Epoch 12, CIFAR-10 Batch 4:  Loss: 1.0288  Validation Accuracy: 0.5810
Epoch 12, CIFAR-10 Batch 5:  Loss: 0.8717  Validation Accuracy: 0.5870
Epoch 13, CIFAR-10 Batch 1:  Loss: 1.0185  Validation Accuracy: 0.5616
```

```
Epoch 13, CIFAR-10 Batch 2:  Loss: 0.8769  Validation Accuracy: 0.5888
Epoch 13, CIFAR-10 Batch 3:  Loss: 0.8673  Validation Accuracy: 0.5710
Epoch 13, CIFAR-10 Batch 4:  Loss: 0.9661  Validation Accuracy: 0.5870
Epoch 13, CIFAR-10 Batch 5:  Loss: 0.8071  Validation Accuracy: 0.5940
Epoch 14, CIFAR-10 Batch 1:  Loss: 0.9513  Validation Accuracy: 0.5686
Epoch 14, CIFAR-10 Batch 2:  Loss: 0.8549  Validation Accuracy: 0.5882
Epoch 14, CIFAR-10 Batch 3:  Loss: 0.7269  Validation Accuracy: 0.5890
Epoch 14, CIFAR-10 Batch 4:  Loss: 0.9111  Validation Accuracy: 0.5958
Epoch 14, CIFAR-10 Batch 5:  Loss: 0.7728  Validation Accuracy: 0.6014
Epoch 15, CIFAR-10 Batch 1:  Loss: 0.9081  Validation Accuracy: 0.5788
Epoch 15, CIFAR-10 Batch 2:  Loss: 0.7984  Validation Accuracy: 0.5886
Epoch 15, CIFAR-10 Batch 3:  Loss: 0.7086  Validation Accuracy: 0.5924
Epoch 15, CIFAR-10 Batch 4:  Loss: 0.9115  Validation Accuracy: 0.5930
Epoch 15, CIFAR-10 Batch 5:  Loss: 0.7204  Validation Accuracy: 0.6058
Epoch 16, CIFAR-10 Batch 1:  Loss: 0.9075  Validation Accuracy: 0.5750
Epoch 16, CIFAR-10 Batch 2:  Loss: 0.7984  Validation Accuracy: 0.5966
Epoch 16, CIFAR-10 Batch 3:  Loss: 0.6322  Validation Accuracy: 0.6044
Epoch 16, CIFAR-10 Batch 4:  Loss: 0.8999  Validation Accuracy: 0.5962
Epoch 16, CIFAR-10 Batch 5:  Loss: 0.6865  Validation Accuracy: 0.6122
Epoch 17, CIFAR-10 Batch 1:  Loss: 0.8466  Validation Accuracy: 0.5810
Epoch 17, CIFAR-10 Batch 2:  Loss: 0.7507  Validation Accuracy: 0.5976
Epoch 17, CIFAR-10 Batch 3:  Loss: 0.6670  Validation Accuracy: 0.5914
Epoch 17, CIFAR-10 Batch 4:  Loss: 0.7901  Validation Accuracy: 0.6078
Epoch 17, CIFAR-10 Batch 5:  Loss: 0.6369  Validation Accuracy: 0.6048
Epoch 18, CIFAR-10 Batch 1:  Loss: 0.8253  Validation Accuracy: 0.5966
Epoch 18, CIFAR-10 Batch 2:  Loss: 0.7154  Validation Accuracy: 0.6024
Epoch 18, CIFAR-10 Batch 3:  Loss: 0.6291  Validation Accuracy: 0.6072
Epoch 18, CIFAR-10 Batch 4:  Loss: 0.8043  Validation Accuracy: 0.6096
Epoch 18, CIFAR-10 Batch 5:  Loss: 0.6249  Validation Accuracy: 0.6148
Epoch 19, CIFAR-10 Batch 1:  Loss: 0.7903  Validation Accuracy: 0.6060
Epoch 19, CIFAR-10 Batch 2:  Loss: 0.7113  Validation Accuracy: 0.6032
Epoch 19, CIFAR-10 Batch 3:  Loss: 0.5891  Validation Accuracy: 0.6060
Epoch 19, CIFAR-10 Batch 4:  Loss: 0.7701  Validation Accuracy: 0.6116
Epoch 19, CIFAR-10 Batch 5:  Loss: 0.5846  Validation Accuracy: 0.6148
Epoch 20, CIFAR-10 Batch 1:  Loss: 0.7700  Validation Accuracy: 0.6118
Epoch 20, CIFAR-10 Batch 2:  Loss: 0.6895  Validation Accuracy: 0.6082
Epoch 20, CIFAR-10 Batch 3:  Loss: 0.5449  Validation Accuracy: 0.6106
Epoch 20, CIFAR-10 Batch 4:  Loss: 0.7440  Validation Accuracy: 0.6232
Epoch 20, CIFAR-10 Batch 5:  Loss: 0.5239  Validation Accuracy: 0.6306
Epoch 21, CIFAR-10 Batch 1:  Loss: 0.7096  Validation Accuracy: 0.6192
Epoch 21, CIFAR-10 Batch 2:  Loss: 0.6403  Validation Accuracy: 0.6226
Epoch 21, CIFAR-10 Batch 3:  Loss: 0.5562  Validation Accuracy: 0.6124
Epoch 21, CIFAR-10 Batch 4:  Loss: 0.6908  Validation Accuracy: 0.6234
Epoch 21, CIFAR-10 Batch 5:  Loss: 0.5232  Validation Accuracy: 0.6326
Epoch 22, CIFAR-10 Batch 1:  Loss: 0.6805  Validation Accuracy: 0.6148
Epoch 22, CIFAR-10 Batch 2:  Loss: 0.6282  Validation Accuracy: 0.6044
Epoch 22, CIFAR-10 Batch 3:  Loss: 0.5308  Validation Accuracy: 0.6270
Epoch 22, CIFAR-10 Batch 4:  Loss: 0.6384  Validation Accuracy: 0.6316
```

```
Epoch 22, CIFAR-10 Batch 5:  Loss: 0.4973  Validation Accuracy: 0.6316
Epoch 23, CIFAR-10 Batch 1:  Loss: 0.6599  Validation Accuracy: 0.6086
Epoch 23, CIFAR-10 Batch 2:  Loss: 0.5832  Validation Accuracy: 0.6258
Epoch 23, CIFAR-10 Batch 3:  Loss: 0.5687  Validation Accuracy: 0.6170
Epoch 23, CIFAR-10 Batch 4:  Loss: 0.6817  Validation Accuracy: 0.6182
Epoch 23, CIFAR-10 Batch 5:  Loss: 0.4585  Validation Accuracy: 0.6320
Epoch 24, CIFAR-10 Batch 1:  Loss: 0.6522  Validation Accuracy: 0.6106
Epoch 24, CIFAR-10 Batch 2:  Loss: 0.6002  Validation Accuracy: 0.6052
Epoch 24, CIFAR-10 Batch 3:  Loss: 0.5057  Validation Accuracy: 0.6228
Epoch 24, CIFAR-10 Batch 4:  Loss: 0.6265  Validation Accuracy: 0.6254
Epoch 24, CIFAR-10 Batch 5:  Loss: 0.4791  Validation Accuracy: 0.6298
Epoch 25, CIFAR-10 Batch 1:  Loss: 0.6356  Validation Accuracy: 0.6148
Epoch 25, CIFAR-10 Batch 2:  Loss: 0.5806  Validation Accuracy: 0.6328
Epoch 25, CIFAR-10 Batch 3:  Loss: 0.5068  Validation Accuracy: 0.6184
Epoch 25, CIFAR-10 Batch 4:  Loss: 0.5947  Validation Accuracy: 0.6224
Epoch 25, CIFAR-10 Batch 5:  Loss: 0.4412  Validation Accuracy: 0.6322
Epoch 26, CIFAR-10 Batch 1:  Loss: 0.6127  Validation Accuracy: 0.6096
Epoch 26, CIFAR-10 Batch 2:  Loss: 0.5621  Validation Accuracy: 0.6232
Epoch 26, CIFAR-10 Batch 3:  Loss: 0.4865  Validation Accuracy: 0.6114
Epoch 26, CIFAR-10 Batch 4:  Loss: 0.6060  Validation Accuracy: 0.6350
Epoch 26, CIFAR-10 Batch 5:  Loss: 0.4132  Validation Accuracy: 0.6396
Epoch 27, CIFAR-10 Batch 1:  Loss: 0.6192  Validation Accuracy: 0.6164
Epoch 27, CIFAR-10 Batch 2:  Loss: 0.5595  Validation Accuracy: 0.6350
Epoch 27, CIFAR-10 Batch 3:  Loss: 0.4382  Validation Accuracy: 0.6314
Epoch 27, CIFAR-10 Batch 4:  Loss: 0.6016  Validation Accuracy: 0.6298
Epoch 27, CIFAR-10 Batch 5:  Loss: 0.4045  Validation Accuracy: 0.6426
Epoch 28, CIFAR-10 Batch 1:  Loss: 0.6030  Validation Accuracy: 0.6110
Epoch 28, CIFAR-10 Batch 2:  Loss: 0.4863  Validation Accuracy: 0.6368
Epoch 28, CIFAR-10 Batch 3:  Loss: 0.4247  Validation Accuracy: 0.6394
Epoch 28, CIFAR-10 Batch 4:  Loss: 0.5546  Validation Accuracy: 0.6358
Epoch 28, CIFAR-10 Batch 5:  Loss: 0.3845  Validation Accuracy: 0.6372
Epoch 29, CIFAR-10 Batch 1:  Loss: 0.5878  Validation Accuracy: 0.6282
Epoch 29, CIFAR-10 Batch 2:  Loss: 0.4981  Validation Accuracy: 0.6434
Epoch 29, CIFAR-10 Batch 3:  Loss: 0.4222  Validation Accuracy: 0.6200
Epoch 29, CIFAR-10 Batch 4:  Loss: 0.5678  Validation Accuracy: 0.6290
Epoch 29, CIFAR-10 Batch 5:  Loss: 0.3806  Validation Accuracy: 0.6408
Epoch 30, CIFAR-10 Batch 1:  Loss: 0.5599  Validation Accuracy: 0.6292
Epoch 30, CIFAR-10 Batch 2:  Loss: 0.4545  Validation Accuracy: 0.6416
Epoch 30, CIFAR-10 Batch 3:  Loss: 0.3705  Validation Accuracy: 0.6382
Epoch 30, CIFAR-10 Batch 4:  Loss: 0.5332  Validation Accuracy: 0.6250
Epoch 30, CIFAR-10 Batch 5:  Loss: 0.3756  Validation Accuracy: 0.6470
Epoch 31, CIFAR-10 Batch 1:  Loss: 0.5463  Validation Accuracy: 0.6372
Epoch 31, CIFAR-10 Batch 2:  Loss: 0.4421  Validation Accuracy: 0.6290
Epoch 31, CIFAR-10 Batch 3:  Loss: 0.3968  Validation Accuracy: 0.6372
Epoch 31, CIFAR-10 Batch 4:  Loss: 0.5056  Validation Accuracy: 0.6358
Epoch 31, CIFAR-10 Batch 5:  Loss: 0.3685  Validation Accuracy: 0.6388
Epoch 32, CIFAR-10 Batch 1:  Loss: 0.5519  Validation Accuracy: 0.6378
Epoch 32, CIFAR-10 Batch 2:  Loss: 0.4468  Validation Accuracy: 0.6344
```

```
Epoch 32, CIFAR-10 Batch 3:  Loss: 0.3848  Validation Accuracy: 0.6244
Epoch 32, CIFAR-10 Batch 4:  Loss: 0.4847  Validation Accuracy: 0.6336
Epoch 32, CIFAR-10 Batch 5:  Loss: 0.3616  Validation Accuracy: 0.6462
Epoch 33, CIFAR-10 Batch 1:  Loss: 0.5571  Validation Accuracy: 0.6188
Epoch 33, CIFAR-10 Batch 2:  Loss: 0.4692  Validation Accuracy: 0.6280
Epoch 33, CIFAR-10 Batch 3:  Loss: 0.3964  Validation Accuracy: 0.6068
Epoch 33, CIFAR-10 Batch 4:  Loss: 0.4881  Validation Accuracy: 0.6304
Epoch 33, CIFAR-10 Batch 5:  Loss: 0.3554  Validation Accuracy: 0.6466
Epoch 34, CIFAR-10 Batch 1:  Loss: 0.4817  Validation Accuracy: 0.6322
Epoch 34, CIFAR-10 Batch 2:  Loss: 0.4474  Validation Accuracy: 0.6322
Epoch 34, CIFAR-10 Batch 3:  Loss: 0.3523  Validation Accuracy: 0.6366
Epoch 34, CIFAR-10 Batch 4:  Loss: 0.4233  Validation Accuracy: 0.6356
Epoch 34, CIFAR-10 Batch 5:  Loss: 0.3350  Validation Accuracy: 0.6522
Epoch 35, CIFAR-10 Batch 1:  Loss: 0.4697  Validation Accuracy: 0.6370
Epoch 35, CIFAR-10 Batch 2:  Loss: 0.4731  Validation Accuracy: 0.6316
Epoch 35, CIFAR-10 Batch 3:  Loss: 0.3371  Validation Accuracy: 0.6344
Epoch 35, CIFAR-10 Batch 4:  Loss: 0.5013  Validation Accuracy: 0.6062
Epoch 35, CIFAR-10 Batch 5:  Loss: 0.3222  Validation Accuracy: 0.6386
Epoch 36, CIFAR-10 Batch 1:  Loss: 0.4564  Validation Accuracy: 0.6292
Epoch 36, CIFAR-10 Batch 2:  Loss: 0.4307  Validation Accuracy: 0.6324
Epoch 36, CIFAR-10 Batch 3:  Loss: 0.3396  Validation Accuracy: 0.6132
Epoch 36, CIFAR-10 Batch 4:  Loss: 0.4674  Validation Accuracy: 0.6226
Epoch 36, CIFAR-10 Batch 5:  Loss: 0.3108  Validation Accuracy: 0.6438
Epoch 37, CIFAR-10 Batch 1:  Loss: 0.4615  Validation Accuracy: 0.6274
Epoch 37, CIFAR-10 Batch 2:  Loss: 0.4311  Validation Accuracy: 0.6306
Epoch 37, CIFAR-10 Batch 3:  Loss: 0.3537  Validation Accuracy: 0.6186
Epoch 37, CIFAR-10 Batch 4:  Loss: 0.4242  Validation Accuracy: 0.6342
Epoch 37, CIFAR-10 Batch 5:  Loss: 0.3075  Validation Accuracy: 0.6486
Epoch 38, CIFAR-10 Batch 1:  Loss: 0.4387  Validation Accuracy: 0.6364
Epoch 38, CIFAR-10 Batch 2:  Loss: 0.4072  Validation Accuracy: 0.6364
Epoch 38, CIFAR-10 Batch 3:  Loss: 0.3738  Validation Accuracy: 0.5998
Epoch 38, CIFAR-10 Batch 4:  Loss: 0.4240  Validation Accuracy: 0.6334
Epoch 38, CIFAR-10 Batch 5:  Loss: 0.3304  Validation Accuracy: 0.6458
Epoch 39, CIFAR-10 Batch 1:  Loss: 0.4379  Validation Accuracy: 0.6418
Epoch 39, CIFAR-10 Batch 2:  Loss: 0.3932  Validation Accuracy: 0.6472
Epoch 39, CIFAR-10 Batch 3:  Loss: 0.3375  Validation Accuracy: 0.6394
Epoch 39, CIFAR-10 Batch 4:  Loss: 0.4067  Validation Accuracy: 0.6474
Epoch 39, CIFAR-10 Batch 5:  Loss: 0.2993  Validation Accuracy: 0.6454
Epoch 40, CIFAR-10 Batch 1:  Loss: 0.4051  Validation Accuracy: 0.6432
Epoch 40, CIFAR-10 Batch 2:  Loss: 0.3624  Validation Accuracy: 0.6428
Epoch 40, CIFAR-10 Batch 3:  Loss: 0.3075  Validation Accuracy: 0.6276
Epoch 40, CIFAR-10 Batch 4:  Loss: 0.3944  Validation Accuracy: 0.6370
Epoch 40, CIFAR-10 Batch 5:  Loss: 0.2922  Validation Accuracy: 0.6486
Epoch 41, CIFAR-10 Batch 1:  Loss: 0.4176  Validation Accuracy: 0.6352
Epoch 41, CIFAR-10 Batch 2:  Loss: 0.3518  Validation Accuracy: 0.6526
Epoch 41, CIFAR-10 Batch 3:  Loss: 0.3209  Validation Accuracy: 0.6462
Epoch 41, CIFAR-10 Batch 4:  Loss: 0.3911  Validation Accuracy: 0.6422
Epoch 41, CIFAR-10 Batch 5:  Loss: 0.3045  Validation Accuracy: 0.6450
```

```
Epoch 42, CIFAR-10 Batch 1:  Loss: 0.3912  Validation Accuracy: 0.6416
Epoch 42, CIFAR-10 Batch 2:  Loss: 0.3448  Validation Accuracy: 0.6500
Epoch 42, CIFAR-10 Batch 3:  Loss: 0.2905  Validation Accuracy: 0.6374
Epoch 42, CIFAR-10 Batch 4:  Loss: 0.3845  Validation Accuracy: 0.6428
Epoch 42, CIFAR-10 Batch 5:  Loss: 0.2831  Validation Accuracy: 0.6468
Epoch 43, CIFAR-10 Batch 1:  Loss: 0.3866  Validation Accuracy: 0.6466
Epoch 43, CIFAR-10 Batch 2:  Loss: 0.3314  Validation Accuracy: 0.6296
Epoch 43, CIFAR-10 Batch 3:  Loss: 0.3027  Validation Accuracy: 0.6252
Epoch 43, CIFAR-10 Batch 4:  Loss: 0.3716  Validation Accuracy: 0.6330
Epoch 43, CIFAR-10 Batch 5:  Loss: 0.2905  Validation Accuracy: 0.6324
Epoch 44, CIFAR-10 Batch 1:  Loss: 0.3709  Validation Accuracy: 0.6348
Epoch 44, CIFAR-10 Batch 2:  Loss: 0.3476  Validation Accuracy: 0.6312
Epoch 44, CIFAR-10 Batch 3:  Loss: 0.2899  Validation Accuracy: 0.6252
Epoch 44, CIFAR-10 Batch 4:  Loss: 0.3553  Validation Accuracy: 0.6296
Epoch 44, CIFAR-10 Batch 5:  Loss: 0.2717  Validation Accuracy: 0.6392
Epoch 45, CIFAR-10 Batch 1:  Loss: 0.3694  Validation Accuracy: 0.6360
Epoch 45, CIFAR-10 Batch 2:  Loss: 0.3413  Validation Accuracy: 0.6304
Epoch 45, CIFAR-10 Batch 3:  Loss: 0.2766  Validation Accuracy: 0.6414
Epoch 45, CIFAR-10 Batch 4:  Loss: 0.3524  Validation Accuracy: 0.6304
Epoch 45, CIFAR-10 Batch 5:  Loss: 0.2908  Validation Accuracy: 0.6438
Epoch 46, CIFAR-10 Batch 1:  Loss: 0.3792  Validation Accuracy: 0.6404
Epoch 46, CIFAR-10 Batch 2:  Loss: 0.3384  Validation Accuracy: 0.6344
Epoch 46, CIFAR-10 Batch 3:  Loss: 0.2679  Validation Accuracy: 0.6324
Epoch 46, CIFAR-10 Batch 4:  Loss: 0.3421  Validation Accuracy: 0.6396
Epoch 46, CIFAR-10 Batch 5:  Loss: 0.2892  Validation Accuracy: 0.6428
Epoch 47, CIFAR-10 Batch 1:  Loss: 0.3916  Validation Accuracy: 0.6462
Epoch 47, CIFAR-10 Batch 2:  Loss: 0.3372  Validation Accuracy: 0.6394
Epoch 47, CIFAR-10 Batch 3:  Loss: 0.2645  Validation Accuracy: 0.6396
Epoch 47, CIFAR-10 Batch 4:  Loss: 0.3455  Validation Accuracy: 0.6540
Epoch 47, CIFAR-10 Batch 5:  Loss: 0.2777  Validation Accuracy: 0.6320
Epoch 48, CIFAR-10 Batch 1:  Loss: 0.3612  Validation Accuracy: 0.6544
Epoch 48, CIFAR-10 Batch 2:  Loss: 0.3094  Validation Accuracy: 0.6454
Epoch 48, CIFAR-10 Batch 3:  Loss: 0.2499  Validation Accuracy: 0.6562
Epoch 48, CIFAR-10 Batch 4:  Loss: 0.3579  Validation Accuracy: 0.6436
Epoch 48, CIFAR-10 Batch 5:  Loss: 0.2731  Validation Accuracy: 0.6492
Epoch 49, CIFAR-10 Batch 1:  Loss: 0.3780  Validation Accuracy: 0.6418
Epoch 49, CIFAR-10 Batch 2:  Loss: 0.3005  Validation Accuracy: 0.6324
Epoch 49, CIFAR-10 Batch 3:  Loss: 0.2516  Validation Accuracy: 0.6512
Epoch 49, CIFAR-10 Batch 4:  Loss: 0.3675  Validation Accuracy: 0.6340
Epoch 49, CIFAR-10 Batch 5:  Loss: 0.2883  Validation Accuracy: 0.6420
Epoch 50, CIFAR-10 Batch 1:  Loss: 0.4280  Validation Accuracy: 0.6308
Epoch 50, CIFAR-10 Batch 2:  Loss: 0.3101  Validation Accuracy: 0.6422
Epoch 50, CIFAR-10 Batch 3:  Loss: 0.2526  Validation Accuracy: 0.6554
Epoch 50, CIFAR-10 Batch 4:  Loss: 0.3487  Validation Accuracy: 0.6348
Epoch 50, CIFAR-10 Batch 5:  Loss: 0.2617  Validation Accuracy: 0.6418
Epoch 51, CIFAR-10 Batch 1:  Loss: 0.3641  Validation Accuracy: 0.6394
Epoch 51, CIFAR-10 Batch 2:  Loss: 0.2870  Validation Accuracy: 0.6398
Epoch 51, CIFAR-10 Batch 3:  Loss: 0.2411  Validation Accuracy: 0.6546
```

```
Epoch 51, CIFAR-10 Batch 4:  Loss: 0.3327  Validation Accuracy: 0.6348
Epoch 51, CIFAR-10 Batch 5:  Loss: 0.2501  Validation Accuracy: 0.6488
Epoch 52, CIFAR-10 Batch 1:  Loss: 0.3731  Validation Accuracy: 0.6390
Epoch 52, CIFAR-10 Batch 2:  Loss: 0.2928  Validation Accuracy: 0.6406
Epoch 52, CIFAR-10 Batch 3:  Loss: 0.2405  Validation Accuracy: 0.6568
Epoch 52, CIFAR-10 Batch 4:  Loss: 0.3327  Validation Accuracy: 0.6334
Epoch 52, CIFAR-10 Batch 5:  Loss: 0.2160  Validation Accuracy: 0.6474
Epoch 53, CIFAR-10 Batch 1:  Loss: 0.3493  Validation Accuracy: 0.6344
Epoch 53, CIFAR-10 Batch 2:  Loss: 0.2666  Validation Accuracy: 0.6428
Epoch 53, CIFAR-10 Batch 3:  Loss: 0.2388  Validation Accuracy: 0.6454
Epoch 53, CIFAR-10 Batch 4:  Loss: 0.3200  Validation Accuracy: 0.6378
Epoch 53, CIFAR-10 Batch 5:  Loss: 0.2300  Validation Accuracy: 0.6528
Epoch 54, CIFAR-10 Batch 1:  Loss: 0.3733  Validation Accuracy: 0.6178
Epoch 54, CIFAR-10 Batch 2:  Loss: 0.3019  Validation Accuracy: 0.6290
Epoch 54, CIFAR-10 Batch 3:  Loss: 0.2400  Validation Accuracy: 0.6376
Epoch 54, CIFAR-10 Batch 4:  Loss: 0.3577  Validation Accuracy: 0.6276
Epoch 54, CIFAR-10 Batch 5:  Loss: 0.2236  Validation Accuracy: 0.6530
Epoch 55, CIFAR-10 Batch 1:  Loss: 0.3323  Validation Accuracy: 0.6360
Epoch 55, CIFAR-10 Batch 2:  Loss: 0.2600  Validation Accuracy: 0.6308
Epoch 55, CIFAR-10 Batch 3:  Loss: 0.2206  Validation Accuracy: 0.6520
Epoch 55, CIFAR-10 Batch 4:  Loss: 0.3429  Validation Accuracy: 0.6312
Epoch 55, CIFAR-10 Batch 5:  Loss: 0.2190  Validation Accuracy: 0.6502
Epoch 56, CIFAR-10 Batch 1:  Loss: 0.3055  Validation Accuracy: 0.6400
Epoch 56, CIFAR-10 Batch 2:  Loss: 0.2594  Validation Accuracy: 0.6422
Epoch 56, CIFAR-10 Batch 3:  Loss: 0.2274  Validation Accuracy: 0.6526
Epoch 56, CIFAR-10 Batch 4:  Loss: 0.3353  Validation Accuracy: 0.6284
Epoch 56, CIFAR-10 Batch 5:  Loss: 0.2343  Validation Accuracy: 0.6504
Epoch 57, CIFAR-10 Batch 1:  Loss: 0.3137  Validation Accuracy: 0.6350
Epoch 57, CIFAR-10 Batch 2:  Loss: 0.2665  Validation Accuracy: 0.6426
Epoch 57, CIFAR-10 Batch 3:  Loss: 0.2312  Validation Accuracy: 0.6540
Epoch 57, CIFAR-10 Batch 4:  Loss: 0.2990  Validation Accuracy: 0.6476
Epoch 57, CIFAR-10 Batch 5:  Loss: 0.2097  Validation Accuracy: 0.6534
Epoch 58, CIFAR-10 Batch 1:  Loss: 0.3075  Validation Accuracy: 0.6314
Epoch 58, CIFAR-10 Batch 2:  Loss: 0.2558  Validation Accuracy: 0.6470
Epoch 58, CIFAR-10 Batch 3:  Loss: 0.2645  Validation Accuracy: 0.6482
Epoch 58, CIFAR-10 Batch 4:  Loss: 0.3015  Validation Accuracy: 0.6478
Epoch 58, CIFAR-10 Batch 5:  Loss: 0.2121  Validation Accuracy: 0.6438
Epoch 59, CIFAR-10 Batch 1:  Loss: 0.3046  Validation Accuracy: 0.6440
Epoch 59, CIFAR-10 Batch 2:  Loss: 0.2689  Validation Accuracy: 0.6296
Epoch 59, CIFAR-10 Batch 3:  Loss: 0.2326  Validation Accuracy: 0.6476
Epoch 59, CIFAR-10 Batch 4:  Loss: 0.3182  Validation Accuracy: 0.6344
Epoch 59, CIFAR-10 Batch 5:  Loss: 0.2111  Validation Accuracy: 0.6408
Epoch 60, CIFAR-10 Batch 1:  Loss: 0.3246  Validation Accuracy: 0.6378
Epoch 60, CIFAR-10 Batch 2:  Loss: 0.2648  Validation Accuracy: 0.6424
Epoch 60, CIFAR-10 Batch 3:  Loss: 0.2256  Validation Accuracy: 0.6538
Epoch 60, CIFAR-10 Batch 4:  Loss: 0.3239  Validation Accuracy: 0.6210
Epoch 60, CIFAR-10 Batch 5:  Loss: 0.2322  Validation Accuracy: 0.6274
Epoch 61, CIFAR-10 Batch 1:  Loss: 0.2865  Validation Accuracy: 0.6496
```

```
Epoch 61, CIFAR-10 Batch 2:  Loss: 0.2661  Validation Accuracy: 0.6438
Epoch 61, CIFAR-10 Batch 3:  Loss: 0.2210  Validation Accuracy: 0.6512
Epoch 61, CIFAR-10 Batch 4:  Loss: 0.3262  Validation Accuracy: 0.6090
Epoch 61, CIFAR-10 Batch 5:  Loss: 0.2232  Validation Accuracy: 0.6228
Epoch 62, CIFAR-10 Batch 1:  Loss: 0.2834  Validation Accuracy: 0.6456
Epoch 62, CIFAR-10 Batch 2:  Loss: 0.2854  Validation Accuracy: 0.6276
Epoch 62, CIFAR-10 Batch 3:  Loss: 0.2391  Validation Accuracy: 0.6516
Epoch 62, CIFAR-10 Batch 4:  Loss: 0.3325  Validation Accuracy: 0.6140
Epoch 62, CIFAR-10 Batch 5:  Loss: 0.2150  Validation Accuracy: 0.6402
Epoch 63, CIFAR-10 Batch 1:  Loss: 0.2650  Validation Accuracy: 0.6428
Epoch 63, CIFAR-10 Batch 2:  Loss: 0.2696  Validation Accuracy: 0.6236
Epoch 63, CIFAR-10 Batch 3:  Loss: 0.2413  Validation Accuracy: 0.6490
Epoch 63, CIFAR-10 Batch 4:  Loss: 0.2805  Validation Accuracy: 0.6308
Epoch 63, CIFAR-10 Batch 5:  Loss: 0.2113  Validation Accuracy: 0.6400
Epoch 64, CIFAR-10 Batch 1:  Loss: 0.2712  Validation Accuracy: 0.6426
Epoch 64, CIFAR-10 Batch 2:  Loss: 0.2723  Validation Accuracy: 0.6206
Epoch 64, CIFAR-10 Batch 3:  Loss: 0.2622  Validation Accuracy: 0.6412
Epoch 64, CIFAR-10 Batch 4:  Loss: 0.3032  Validation Accuracy: 0.6282
Epoch 64, CIFAR-10 Batch 5:  Loss: 0.2025  Validation Accuracy: 0.6464
Epoch 65, CIFAR-10 Batch 1:  Loss: 0.2795  Validation Accuracy: 0.6376
Epoch 65, CIFAR-10 Batch 2:  Loss: 0.2489  Validation Accuracy: 0.6264
Epoch 65, CIFAR-10 Batch 3:  Loss: 0.2479  Validation Accuracy: 0.6322
Epoch 65, CIFAR-10 Batch 4:  Loss: 0.2795  Validation Accuracy: 0.6378
Epoch 65, CIFAR-10 Batch 5:  Loss: 0.2124  Validation Accuracy: 0.6358
Epoch 66, CIFAR-10 Batch 1:  Loss: 0.2587  Validation Accuracy: 0.6452
Epoch 66, CIFAR-10 Batch 2:  Loss: 0.2988  Validation Accuracy: 0.6108
Epoch 66, CIFAR-10 Batch 3:  Loss: 0.2073  Validation Accuracy: 0.6386
Epoch 66, CIFAR-10 Batch 4:  Loss: 0.2663  Validation Accuracy: 0.6330
Epoch 66, CIFAR-10 Batch 5:  Loss: 0.2007  Validation Accuracy: 0.6486
Epoch 67, CIFAR-10 Batch 1:  Loss: 0.2546  Validation Accuracy: 0.6434
Epoch 67, CIFAR-10 Batch 2:  Loss: 0.2880  Validation Accuracy: 0.6150
Epoch 67, CIFAR-10 Batch 3:  Loss: 0.2007  Validation Accuracy: 0.6262
Epoch 67, CIFAR-10 Batch 4:  Loss: 0.2541  Validation Accuracy: 0.6290
Epoch 67, CIFAR-10 Batch 5:  Loss: 0.2021  Validation Accuracy: 0.6390
Epoch 68, CIFAR-10 Batch 1:  Loss: 0.2647  Validation Accuracy: 0.6356
Epoch 68, CIFAR-10 Batch 2:  Loss: 0.2966  Validation Accuracy: 0.6036
Epoch 68, CIFAR-10 Batch 3:  Loss: 0.2285  Validation Accuracy: 0.6198
Epoch 68, CIFAR-10 Batch 4:  Loss: 0.2439  Validation Accuracy: 0.6390
Epoch 68, CIFAR-10 Batch 5:  Loss: 0.2083  Validation Accuracy: 0.6400
Epoch 69, CIFAR-10 Batch 1:  Loss: 0.2751  Validation Accuracy: 0.6436
Epoch 69, CIFAR-10 Batch 2:  Loss: 0.2694  Validation Accuracy: 0.6266
Epoch 69, CIFAR-10 Batch 3:  Loss: 0.2105  Validation Accuracy: 0.6280
Epoch 69, CIFAR-10 Batch 4:  Loss: 0.2398  Validation Accuracy: 0.6422
Epoch 69, CIFAR-10 Batch 5:  Loss: 0.1950  Validation Accuracy: 0.6372
Epoch 70, CIFAR-10 Batch 1:  Loss: 0.2716  Validation Accuracy: 0.6420
Epoch 70, CIFAR-10 Batch 2:  Loss: 0.2436  Validation Accuracy: 0.6402
Epoch 70, CIFAR-10 Batch 3:  Loss: 0.2004  Validation Accuracy: 0.6508
Epoch 70, CIFAR-10 Batch 4:  Loss: 0.2300  Validation Accuracy: 0.6410
```

```
Epoch 70, CIFAR-10 Batch 5:  Loss: 0.1930  Validation Accuracy: 0.6328
Epoch 71, CIFAR-10 Batch 1:  Loss: 0.2587  Validation Accuracy: 0.6426
Epoch 71, CIFAR-10 Batch 2:  Loss: 0.2366  Validation Accuracy: 0.6370
Epoch 71, CIFAR-10 Batch 3:  Loss: 0.1911  Validation Accuracy: 0.6486
Epoch 71, CIFAR-10 Batch 4:  Loss: 0.2198  Validation Accuracy: 0.6358
Epoch 71, CIFAR-10 Batch 5:  Loss: 0.1809  Validation Accuracy: 0.6346
Epoch 72, CIFAR-10 Batch 1:  Loss: 0.2598  Validation Accuracy: 0.6372
Epoch 72, CIFAR-10 Batch 2:  Loss: 0.2436  Validation Accuracy: 0.6452
Epoch 72, CIFAR-10 Batch 3:  Loss: 0.2185  Validation Accuracy: 0.6304
Epoch 72, CIFAR-10 Batch 4:  Loss: 0.2427  Validation Accuracy: 0.6314
Epoch 72, CIFAR-10 Batch 5:  Loss: 0.1732  Validation Accuracy: 0.6368
Epoch 73, CIFAR-10 Batch 1:  Loss: 0.2552  Validation Accuracy: 0.6416
Epoch 73, CIFAR-10 Batch 2:  Loss: 0.2245  Validation Accuracy: 0.6466
Epoch 73, CIFAR-10 Batch 3:  Loss: 0.1961  Validation Accuracy: 0.6438
Epoch 73, CIFAR-10 Batch 4:  Loss: 0.2209  Validation Accuracy: 0.6416
Epoch 73, CIFAR-10 Batch 5:  Loss: 0.1814  Validation Accuracy: 0.6326
Epoch 74, CIFAR-10 Batch 1:  Loss: 0.2662  Validation Accuracy: 0.6360
Epoch 74, CIFAR-10 Batch 2:  Loss: 0.2216  Validation Accuracy: 0.6350
Epoch 74, CIFAR-10 Batch 3:  Loss: 0.1966  Validation Accuracy: 0.6484
Epoch 74, CIFAR-10 Batch 4:  Loss: 0.2557  Validation Accuracy: 0.6176
Epoch 74, CIFAR-10 Batch 5:  Loss: 0.1733  Validation Accuracy: 0.6292
Epoch 75, CIFAR-10 Batch 1:  Loss: 0.2501  Validation Accuracy: 0.6418
Epoch 75, CIFAR-10 Batch 2:  Loss: 0.2571  Validation Accuracy: 0.6328
Epoch 75, CIFAR-10 Batch 3:  Loss: 0.2178  Validation Accuracy: 0.6292
Epoch 75, CIFAR-10 Batch 4:  Loss: 0.2362  Validation Accuracy: 0.6288
Epoch 75, CIFAR-10 Batch 5:  Loss: 0.1634  Validation Accuracy: 0.6386
Epoch 76, CIFAR-10 Batch 1:  Loss: 0.2718  Validation Accuracy: 0.6322
Epoch 76, CIFAR-10 Batch 2:  Loss: 0.2238  Validation Accuracy: 0.6388
Epoch 76, CIFAR-10 Batch 3:  Loss: 0.1720  Validation Accuracy: 0.6512
Epoch 76, CIFAR-10 Batch 4:  Loss: 0.2498  Validation Accuracy: 0.6192
Epoch 76, CIFAR-10 Batch 5:  Loss: 0.1589  Validation Accuracy: 0.6454
Epoch 77, CIFAR-10 Batch 1:  Loss: 0.2742  Validation Accuracy: 0.6272
Epoch 77, CIFAR-10 Batch 2:  Loss: 0.2427  Validation Accuracy: 0.6346
Epoch 77, CIFAR-10 Batch 3:  Loss: 0.1961  Validation Accuracy: 0.6418
Epoch 77, CIFAR-10 Batch 4:  Loss: 0.2515  Validation Accuracy: 0.6248
Epoch 77, CIFAR-10 Batch 5:  Loss: 0.1556  Validation Accuracy: 0.6414
Epoch 78, CIFAR-10 Batch 1:  Loss: 0.2605  Validation Accuracy: 0.6272
Epoch 78, CIFAR-10 Batch 2:  Loss: 0.2515  Validation Accuracy: 0.6420
Epoch 78, CIFAR-10 Batch 3:  Loss: 0.1898  Validation Accuracy: 0.6500
Epoch 78, CIFAR-10 Batch 4:  Loss: 0.2189  Validation Accuracy: 0.6356
Epoch 78, CIFAR-10 Batch 5:  Loss: 0.1661  Validation Accuracy: 0.6428
Epoch 79, CIFAR-10 Batch 1:  Loss: 0.2716  Validation Accuracy: 0.6246
Epoch 79, CIFAR-10 Batch 2:  Loss: 0.2443  Validation Accuracy: 0.6450
Epoch 79, CIFAR-10 Batch 3:  Loss: 0.1784  Validation Accuracy: 0.6496
Epoch 79, CIFAR-10 Batch 4:  Loss: 0.2389  Validation Accuracy: 0.6320
Epoch 79, CIFAR-10 Batch 5:  Loss: 0.1790  Validation Accuracy: 0.6336
Epoch 80, CIFAR-10 Batch 1:  Loss: 0.3127  Validation Accuracy: 0.5988
Epoch 80, CIFAR-10 Batch 2:  Loss: 0.2769  Validation Accuracy: 0.6244
```

```
Epoch 80, CIFAR-10 Batch 3:  Loss: 0.1831  Validation Accuracy: 0.6418
Epoch 80, CIFAR-10 Batch 4:  Loss: 0.2196  Validation Accuracy: 0.6372
Epoch 80, CIFAR-10 Batch 5:  Loss: 0.2192  Validation Accuracy: 0.6426
Epoch 81, CIFAR-10 Batch 1:  Loss: 0.3104  Validation Accuracy: 0.6128
Epoch 81, CIFAR-10 Batch 2:  Loss: 0.2602  Validation Accuracy: 0.6324
Epoch 81, CIFAR-10 Batch 3:  Loss: 0.1870  Validation Accuracy: 0.6450
Epoch 81, CIFAR-10 Batch 4:  Loss: 0.2290  Validation Accuracy: 0.6382
Epoch 81, CIFAR-10 Batch 5:  Loss: 0.1880  Validation Accuracy: 0.6394
Epoch 82, CIFAR-10 Batch 1:  Loss: 0.2928  Validation Accuracy: 0.6054
Epoch 82, CIFAR-10 Batch 2:  Loss: 0.2574  Validation Accuracy: 0.6316
Epoch 82, CIFAR-10 Batch 3:  Loss: 0.2122  Validation Accuracy: 0.6394
Epoch 82, CIFAR-10 Batch 4:  Loss: 0.2171  Validation Accuracy: 0.6400
Epoch 82, CIFAR-10 Batch 5:  Loss: 0.1932  Validation Accuracy: 0.6374
Epoch 83, CIFAR-10 Batch 1:  Loss: 0.2793  Validation Accuracy: 0.6216
Epoch 83, CIFAR-10 Batch 2:  Loss: 0.2431  Validation Accuracy: 0.6240
Epoch 83, CIFAR-10 Batch 3:  Loss: 0.1960  Validation Accuracy: 0.6358
Epoch 83, CIFAR-10 Batch 4:  Loss: 0.2049  Validation Accuracy: 0.6428
Epoch 83, CIFAR-10 Batch 5:  Loss: 0.2085  Validation Accuracy: 0.6498
Epoch 84, CIFAR-10 Batch 1:  Loss: 0.2395  Validation Accuracy: 0.6360
Epoch 84, CIFAR-10 Batch 2:  Loss: 0.2303  Validation Accuracy: 0.6360
Epoch 84, CIFAR-10 Batch 3:  Loss: 0.1980  Validation Accuracy: 0.6394
Epoch 84, CIFAR-10 Batch 4:  Loss: 0.2254  Validation Accuracy: 0.6294
Epoch 84, CIFAR-10 Batch 5:  Loss: 0.1802  Validation Accuracy: 0.6502
Epoch 85, CIFAR-10 Batch 1:  Loss: 0.2700  Validation Accuracy: 0.6276
Epoch 85, CIFAR-10 Batch 2:  Loss: 0.2296  Validation Accuracy: 0.6364
Epoch 85, CIFAR-10 Batch 3:  Loss: 0.1832  Validation Accuracy: 0.6418
Epoch 85, CIFAR-10 Batch 4:  Loss: 0.2096  Validation Accuracy: 0.6388
Epoch 85, CIFAR-10 Batch 5:  Loss: 0.1724  Validation Accuracy: 0.6450
Epoch 86, CIFAR-10 Batch 1:  Loss: 0.2597  Validation Accuracy: 0.6400
Epoch 86, CIFAR-10 Batch 2:  Loss: 0.2264  Validation Accuracy: 0.6378
Epoch 86, CIFAR-10 Batch 3:  Loss: 0.1885  Validation Accuracy: 0.6398
Epoch 86, CIFAR-10 Batch 4:  Loss: 0.2105  Validation Accuracy: 0.6424
Epoch 86, CIFAR-10 Batch 5:  Loss: 0.1747  Validation Accuracy: 0.6490
Epoch 87, CIFAR-10 Batch 1:  Loss: 0.2370  Validation Accuracy: 0.6366
Epoch 87, CIFAR-10 Batch 2:  Loss: 0.2093  Validation Accuracy: 0.6366
Epoch 87, CIFAR-10 Batch 3:  Loss: 0.1741  Validation Accuracy: 0.6472
Epoch 87, CIFAR-10 Batch 4:  Loss: 0.1982  Validation Accuracy: 0.6396
Epoch 87, CIFAR-10 Batch 5:  Loss: 0.1822  Validation Accuracy: 0.6462
Epoch 88, CIFAR-10 Batch 1:  Loss: 0.2474  Validation Accuracy: 0.6386
Epoch 88, CIFAR-10 Batch 2:  Loss: 0.1904  Validation Accuracy: 0.6502
Epoch 88, CIFAR-10 Batch 3:  Loss: 0.1778  Validation Accuracy: 0.6440
Epoch 88, CIFAR-10 Batch 4:  Loss: 0.2058  Validation Accuracy: 0.6468
Epoch 88, CIFAR-10 Batch 5:  Loss: 0.1649  Validation Accuracy: 0.6398
Epoch 89, CIFAR-10 Batch 1:  Loss: 0.2460  Validation Accuracy: 0.6386
Epoch 89, CIFAR-10 Batch 2:  Loss: 0.1920  Validation Accuracy: 0.6458
Epoch 89, CIFAR-10 Batch 3:  Loss: 0.1608  Validation Accuracy: 0.6498
Epoch 89, CIFAR-10 Batch 4:  Loss: 0.1980  Validation Accuracy: 0.6404
Epoch 89, CIFAR-10 Batch 5:  Loss: 0.1560  Validation Accuracy: 0.6434
```

```
Epoch 90, CIFAR-10 Batch 1:  Loss: 0.2191  Validation Accuracy: 0.6424
Epoch 90, CIFAR-10 Batch 2:  Loss: 0.1853  Validation Accuracy: 0.6482
Epoch 90, CIFAR-10 Batch 3:  Loss: 0.1737  Validation Accuracy: 0.6468
Epoch 90, CIFAR-10 Batch 4:  Loss: 0.1884  Validation Accuracy: 0.6404
Epoch 90, CIFAR-10 Batch 5:  Loss: 0.1495  Validation Accuracy: 0.6502
Epoch 91, CIFAR-10 Batch 1:  Loss: 0.2226  Validation Accuracy: 0.6442
Epoch 91, CIFAR-10 Batch 2:  Loss: 0.2174  Validation Accuracy: 0.6498
Epoch 91, CIFAR-10 Batch 3:  Loss: 0.1689  Validation Accuracy: 0.6368
Epoch 91, CIFAR-10 Batch 4:  Loss: 0.1676  Validation Accuracy: 0.6434
Epoch 91, CIFAR-10 Batch 5:  Loss: 0.1546  Validation Accuracy: 0.6392
Epoch 92, CIFAR-10 Batch 1:  Loss: 0.2123  Validation Accuracy: 0.6364
Epoch 92, CIFAR-10 Batch 2:  Loss: 0.1678  Validation Accuracy: 0.6388
Epoch 92, CIFAR-10 Batch 3:  Loss: 0.1565  Validation Accuracy: 0.6364
Epoch 92, CIFAR-10 Batch 4:  Loss: 0.1873  Validation Accuracy: 0.6390
Epoch 92, CIFAR-10 Batch 5:  Loss: 0.1732  Validation Accuracy: 0.6428
Epoch 93, CIFAR-10 Batch 1:  Loss: 0.2346  Validation Accuracy: 0.6336
Epoch 93, CIFAR-10 Batch 2:  Loss: 0.1776  Validation Accuracy: 0.6410
Epoch 93, CIFAR-10 Batch 3:  Loss: 0.1882  Validation Accuracy: 0.6382
Epoch 93, CIFAR-10 Batch 4:  Loss: 0.1949  Validation Accuracy: 0.6378
Epoch 93, CIFAR-10 Batch 5:  Loss: 0.1514  Validation Accuracy: 0.6416
Epoch 94, CIFAR-10 Batch 1:  Loss: 0.2407  Validation Accuracy: 0.6344
Epoch 94, CIFAR-10 Batch 2:  Loss: 0.1696  Validation Accuracy: 0.6326
Epoch 94, CIFAR-10 Batch 3:  Loss: 0.1772  Validation Accuracy: 0.6340
Epoch 94, CIFAR-10 Batch 4:  Loss: 0.1968  Validation Accuracy: 0.6410
Epoch 94, CIFAR-10 Batch 5:  Loss: 0.1820  Validation Accuracy: 0.6410
Epoch 95, CIFAR-10 Batch 1:  Loss: 0.2244  Validation Accuracy: 0.6336
Epoch 95, CIFAR-10 Batch 2:  Loss: 0.1742  Validation Accuracy: 0.6320
Epoch 95, CIFAR-10 Batch 3:  Loss: 0.1717  Validation Accuracy: 0.6316
Epoch 95, CIFAR-10 Batch 4:  Loss: 0.1783  Validation Accuracy: 0.6318
Epoch 95, CIFAR-10 Batch 5:  Loss: 0.1520  Validation Accuracy: 0.6384
Epoch 96, CIFAR-10 Batch 1:  Loss: 0.2517  Validation Accuracy: 0.6302
Epoch 96, CIFAR-10 Batch 2:  Loss: 0.1708  Validation Accuracy: 0.6336
Epoch 96, CIFAR-10 Batch 3:  Loss: 0.1789  Validation Accuracy: 0.6248
Epoch 96, CIFAR-10 Batch 4:  Loss: 0.1887  Validation Accuracy: 0.6356
Epoch 96, CIFAR-10 Batch 5:  Loss: 0.1775  Validation Accuracy: 0.6246
Epoch 97, CIFAR-10 Batch 1:  Loss: 0.2304  Validation Accuracy: 0.6290
Epoch 97, CIFAR-10 Batch 2:  Loss: 0.1752  Validation Accuracy: 0.6338
Epoch 97, CIFAR-10 Batch 3:  Loss: 0.1847  Validation Accuracy: 0.6234
Epoch 97, CIFAR-10 Batch 4:  Loss: 0.2100  Validation Accuracy: 0.6440
Epoch 97, CIFAR-10 Batch 5:  Loss: 0.1756  Validation Accuracy: 0.6250
Epoch 98, CIFAR-10 Batch 1:  Loss: 0.2312  Validation Accuracy: 0.6346
Epoch 98, CIFAR-10 Batch 2:  Loss: 0.1651  Validation Accuracy: 0.6380
Epoch 98, CIFAR-10 Batch 3:  Loss: 0.1646  Validation Accuracy: 0.6332
Epoch 98, CIFAR-10 Batch 4:  Loss: 0.2051  Validation Accuracy: 0.6398
Epoch 98, CIFAR-10 Batch 5:  Loss: 0.1639  Validation Accuracy: 0.6170
Epoch 99, CIFAR-10 Batch 1:  Loss: 0.2093  Validation Accuracy: 0.6452
Epoch 99, CIFAR-10 Batch 2:  Loss: 0.1753  Validation Accuracy: 0.6388
Epoch 99, CIFAR-10 Batch 3:  Loss: 0.1700  Validation Accuracy: 0.6346
```

```
Epoch 99, CIFAR-10 Batch 4:  Loss: 0.1869  Validation Accuracy: 0.6384
Epoch 99, CIFAR-10 Batch 5:  Loss: 0.1447  Validation Accuracy: 0.6144
Epoch 100, CIFAR-10 Batch 1:  Loss: 0.2100  Validation Accuracy: 0.6448
Epoch 100, CIFAR-10 Batch 2:  Loss: 0.1642  Validation Accuracy: 0.6296
Epoch 100, CIFAR-10 Batch 3:  Loss: 0.1953  Validation Accuracy: 0.6218
Epoch 100, CIFAR-10 Batch 4:  Loss: 0.1992  Validation Accuracy: 0.6394
Epoch 100, CIFAR-10 Batch 5:  Loss: 0.1527  Validation Accuracy: 0.6220
```

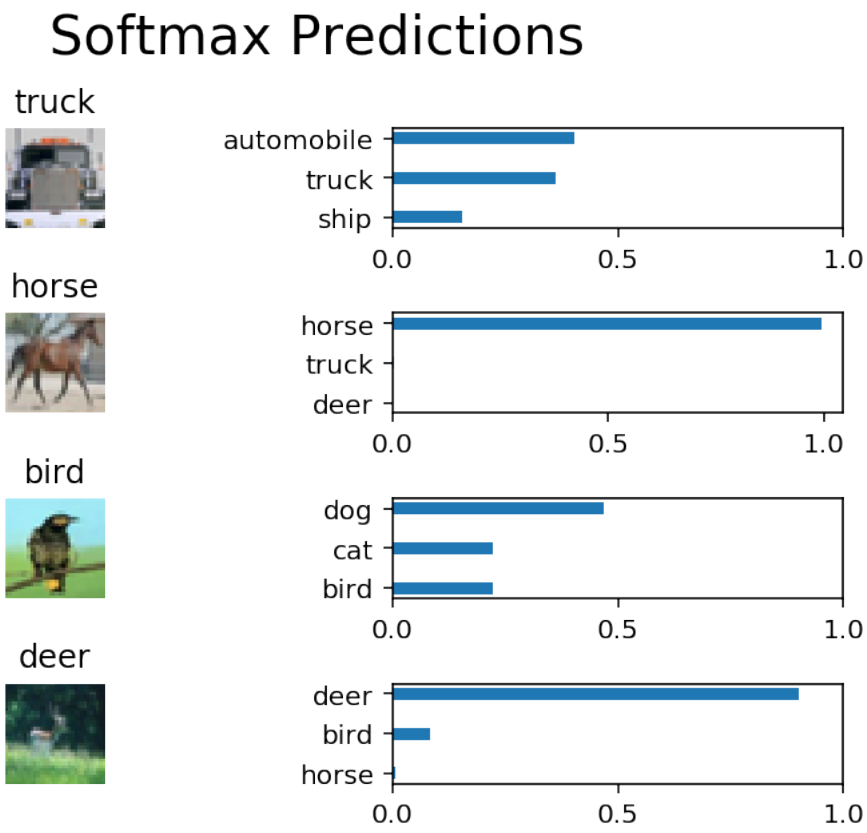
3 Checkpoint

The model has been saved to disk. ## Test Model Test your model against the test dataset. This will be your final accuracy. You should have an accuracy greater than 50%. If you don't, keep tweaking the model architecture and parameters.

In [193]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

import tensorflow as tf
import pickle
import helper
import random

# Set batch size if not already set
try:
    if batch_size:
        pass
except NameError:
    batch_size = 64

save_model_path = './image_classification'
n_samples = 4
top_n_predictions = 3

def test_model():
    """
    Test the saved model against the test dataset
    """

    test_features, test_labels = pickle.load(open('preprocess_training.p', mode='rb'))
    loaded_graph = tf.Graph()

    with tf.Session(graph=loaded_graph) as sess:
        # Load model
```

```
            loader = tf.train.import_meta_graph(save_model_path + '.meta')
            loader.restore(sess, save_model_path)

            # Get Tensors from loaded model
            loaded_x = loaded_graph.get_tensor_by_name('x:0')
            loaded_y = loaded_graph.get_tensor_by_name('y:0')
            loaded_keep_prob = loaded_graph.get_tensor_by_name('keep_prob:0')
            loaded_logits = loaded_graph.get_tensor_by_name('logits:0')
            loaded_acc = loaded_graph.get_tensor_by_name('accuracy:0')

            # Get accuracy in batches for memory limitations
            test_batch_acc_total = 0
            test_batch_count = 0

            for train_feature_batch, train_label_batch in helper.batch_features_labels(tes
                test_batch_acc_total += sess.run(
                    loaded_acc,
                    feed_dict={loaded_x: train_feature_batch, loaded_y: train_label_batch,
                test_batch_count += 1

            print('Testing Accuracy: {}\n'.format(test_batch_acc_total/test_batch_count))

            # Print Random Samples
            random_test_features, random_test_labels = tuple(zip(*random.sample(list(zip(t
            random_test_predictions = sess.run(
                tf.nn.top_k(tf.nn.softmax(loaded_logits), top_n_predictions),
                feed_dict={loaded_x: random_test_features, loaded_y: random_test_labels, l
            helper.display_image_predictions(random_test_features, random_test_labels, ran


    test_model()
```

```
Testing Accuracy: 0.61728515625
```



3.1 Why 50-80% Accuracy?

You might be wondering why you can't get an accuracy any higher. First things first, 50% isn't bad for a simple CNN. Pure guessing would get you 10% accuracy. However, you might notice people are getting scores well above 80%. That's because we haven't taught you all there is to know about neural networks. We still need to cover a few more techniques. ## Submitting This Project When submitting this project, make sure to run all the cells before saving the notebook. Save the notebook file as "dlnd_image_classification.ipynb" and save it as a HTML file under "File" -> "Download as". Include the "helper.py" and "problem_unittests.py" files in your submission.