B093040024
郭晏涵

深度視覺

HW6：Cifar10 classification

notebook 執行過程

將 datasets.zip 載入並解壓縮

Task: Check Code

Please check the implementation of the reshape operation in the `FlattenTransform` class, which can be found in `../exercise_06/exercise_code/data/image_folder_dataset.py`.

```
[4]  os.chdir("/content")

[ ]  !unzip  datasets.zip
```

利用 ImageFolderDataset() 創建 datasets

```
download_url = "https://i2dl.vc.in.tum.de/static/data/cifar10.zip"
i2dl_exercises_path = "/content"#os.path.dirname(os.path.abspath(os.getcwd()))
cifar_root = os.path.join(i2dl_exercises_path, "datasets", "cifar10")
# Use the Cifar10 mean and standard deviation computed in Exercise 3.
cifar_mean = np.array([0.49191375, 0.48235852, 0.44673872])
cifar_std = np.array([0.24706447, 0.24346213, 0.26147554])
# Define all the transforms we will apply on the images when
# retrieving them.
rescale_transform = RescaleTransform()
normalize_transform = NormalizeTransform(
        mean=cifar_mean,
        std=cifar_std
)
flatten_transform = FlattenTransform()
compose_transform = ComposeTransform([rescale_transform, normalize_transform,
                                      flatten_transform])
# Create a train, validation and test dataset.
datasets = {}
for mode in ['train', 'val', 'test']:
        crt_dataset = ImageFolderDataset(
                mode=mode,
                root=cifar_root,
                download_url=download_url,
                transform=compose_transform,
                split={'train': 0.6, 'val': 0.2, 'test': 0.2}
        )
        datasets[mode] = crt_dataset
```

利用 DataLoader() 創建 dataloaders

Then, based on this `Dataset` object, we can construct a `Dataloader` object which samples a random mini-batch of data at once.

```
[ ] # Create a dataloader for each split.
    dataloaders = {}
    for mode in ['train', 'val', 'test']:
            crt_dataloader = DataLoader(
                    dataset=datasets[mode],
                    batch_size=256,
                    shuffle=True,
                    drop_last=True,
            )
            dataloaders[mode] = crt_dataloader
```

Because the `Dataloader` has the `__iter__()` method, we can simply iterate through the batches it produces, like this:

```
for batch in dataloader['train']:
        do_something(batch)
```

印出測試資料

```
#Load the data in a dataset without any transformation
dataset = ImageFolderDataset(
                mode=mode,
                root=cifar_root,
                download_url=download_url,
                split={'train': 0.6, 'val': 0.2, 'test': 0.2},
        )

#Retrieve an image from the dataset and flip it
image = dataset[1]['image']
transform = RandomHorizontalFlip(1)
image_flipped = transform(image)

#Show the two images
plt.figure(figsize = (2,2))
plt.subplot(1, 2, 1)
plt.imshow(image.astype('uint8'))
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(image_flipped.astype('uint8'))
plt.axis('off')
plt.title("Left: Original Image, Right: Flipped image")
plt.show()
```

決定 Layers 個數

1.3 Layers

Now, that the data is prepared, we can discuss the model in which we are feeding the data. In our case the model will be a neural network.

In Exercise 5, you implemented a simple 2-layer neural network that had a hidden size as a parameter:

$$\hat{y} = \sigma(\sigma(xW_1 + b_1)W_2 + b_2)$$

where $\sigma(x)$ was the sigmoid function, x was the input, W_1, W_2 the weight matrices and b_1, b_2 the biases for the two layers.

This is how we used this network:

```
input_size = datasets['train'][0]['image'].shape[0]
model = ClassificationNet(input_size=input_size, hidden_size=512)
```

Note that we updated the `ClassificationNet` from the previous exercise, so that now you can customize more: the number of outputs, the choice of activation function, the hidden size etc. We encourage you to check out the implementation in `exercise_code/networks/classification_net.py`

```
num_layer = 2
reg = 0.1

model = ClassificationNet(activation=Sigmoid(), num_layer=num_layer, reg=reg,
                          num_classes=10)
```

實作 LeakyRelu 和 Tanh

Use this cell to test your implementation of the `LeakyRelu` class:

```
from exercise_code.tests.layer_tests import *
print(LeakyReluTestWrapper()())
```

```
LeakyReluForwardTest passed.
LeakyReluBackwardTest passed.
Congratulations you have passed all the unit tests!!! Tests passed: 2/2
Score: 100/100
You secured a score of :100
```

And this cell to test your implementation of the `Tanh` class:

```
print(TanhTestWrapper()())
```

```
TanhForwardTest passed.
TanhBackwardTest passed.
Congratulations you have passed all the unit tests!!! Tests passed: 2/2
Score: 100/100
You secured a score of :100
```

Congratulations, you now implemented all four different activation functions! These activation layers are now ready to be used when you start building your own network.

LeakyRelu forward

```
    def forward(self, x):
        """
        :param x: Inputs, of any shape

        :return out: Output, of the same shape as x
        :return cache: Cache, for backward computation, of the same shape as x
        """
        outputs = None
        cache = None
        ############################################################################
        # TODO:
        # Implement the forward pass of Relu activation function                   #
        ############################################################################

        outputs = np.array(list(map(lambda x: x if x > 0 else 0, x.ravel()))).reshape(x.shape)
        cache = (x, outputs)

        ############################################################################
        #                                                          END OF YOUR CODE
        ############################################################################
        return outputs, cache
```

L eakyRelu backward (求 gredient)

```
    def backward(self, dout, cache):
        """
        :return: dx: the gradient w.r.t. input X, of the same shape as X
        """
        dx = None
        ############################################################################
        # TODO:
        # Implement the backward pass of Relu activation function                  #
        ############################################################################

        dx = np.array(list(map(lambda x: 1 if x > 0 else 0, cache[1].ravel()))).reshape(cache[1].shape) * dout

        ############################################################################
        #                                                 END OF YOUR CODE
        ############################################################################
        return dx
```

Tanh forward

```
    def forward(self, x):
        """
        :param x: Inputs, of any shape

        :return out: Output, of the same shape as x
        :return cache: Cache, for backward computation, of the same shape as x
        """
        outputs = None
        cache = None
        ############################################################################
        # TODO:
        # Implement the forward pass of LeakyRelu activation function              #
        ############################################################################

        outputs = np.array(list(map(lambda x: x if x > 0 else self.slope * x, x.ravel()))).reshape(x.shape)
        cache = (x, outputs)

        ############################################################################
        #                                                 END OF YOUR CODE
        ############################################################################
        return outputs, cache
```

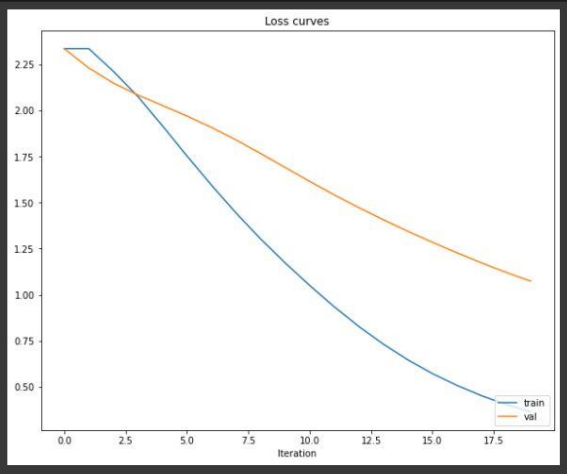
Tanh backward (求 gredient)

```
    def backward(self, dout, cache):
        """
        :return: dx: the gradient w.r.t. input X, of the same shape as X
        """
        dx = None
        ############################################################################
        # TODO:
        # Implement the backward pass of LeakyRelu activation function             #
        ############################################################################

        dx = np.array(list(map(lambda x: 1 if x > 0 else self.slope, cache[1].ravel()))).reshape(cache[1].shape) * dout

        ############################################################################
        #                                                 END OF YOUR CODE
        ############################################################################
        return dx
```

測試 network – part I 層數：2 樣本數：4 reg：0.1

```
[ ] from exercise_code.solver import Solver
    from exercise_code.networks.optimizer import SGD, Adam
    from exercise_code.networks import MyOwnNetwork

    num_layer = 2
    epochs = 20
    reg = 0.1
    batch_size = 4

    model = ClassificationNet(num_layer=num_layer, reg=reg)
    # model = MyOwnNetwork()

    loss = CrossEntropyFromLogits()

    # Make a new data loader with a single training image
    overfit_dataset = ImageFolderDataset(
            mode='train',
            root=cifar_root,
            download_url=download_url,
            transform=compose_transform,
            limit_files=1
    )
    dataloaders['train_overfit_single_image'] = DataLoader(
            dataset=overfit_dataset,
            batch_size=batch_size,
            shuffle=True,
            drop_last=False,
    )

    # Decrease validation data for only debugging
    debugging_validation_dataset = ImageFolderDataset(
            mode='val',
            root=cifar_root,
            download_url=download_url,
            transform=compose_transform,
            limit_files=100
    )
    dataloaders['val_500files'] = DataLoader(
            dataset=debugging_validation_dataset,
            batch_size=batch_size,
            shuffle=True,
            drop_last=True,
    )

    solver = Solver(model, dataloaders['train_overfit_single_image'], dataloaders['val_500files'],
                    learning_rate=1e-3, loss_func=loss, optimizer=Adam)

    solver.train(epochs=epochs)

    (Epoch 1 / 20) train loss: 2.336757; val loss: 2.336790
    (Epoch 2 / 20) train loss: 2.336757; val loss: 2.231507
```

```
    plt.title('Loss curves')
    plt.plot(solver.train_loss_history, '-', label='train')
    plt.plot(solver.val_loss_history, '-', label='val')
    plt.legend(loc='lower right')
    plt.xlabel('Iteration')
    plt.show()
```



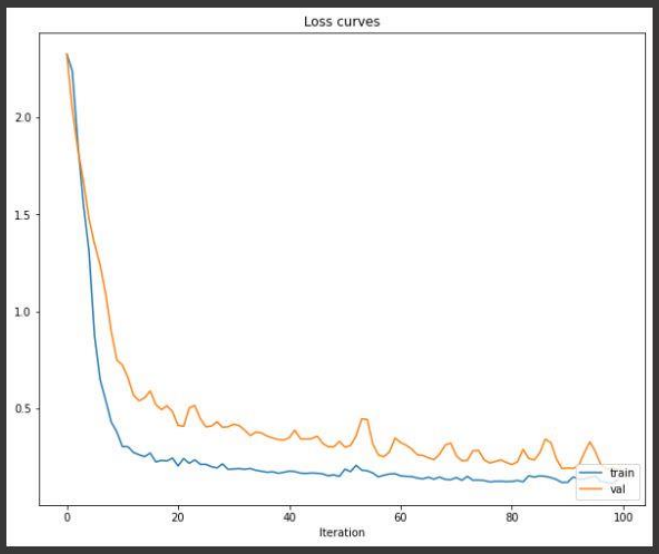

```
[ ] print("Training accuracy: %.5f" % (solver.get_dataset_accuracy(dataloaders['train_overfit_single_image'])))
    print("Validation accuracy: %.5f" % (solver.get_dataset_accuracy(dataloaders['val_500files'])))

    Training accuracy: 1.00000
    Validation accuracy: 1.00000
```

測試 network – part II 層數：2 樣本數：10 reg：0.1

```
from exercise_code.networks import MyOwnNetwork

num_layer = 2
epochs = 100
reg = 0.1
num_samples = 10

model = ClassificationNet(num_layer=num_layer, reg=reg)
# model = MyOwnNetwork()

loss = CrossEntropyFromLogits()

# Make a new data loader with a our num_samples training image
overfit_dataset = ImageFolderDataset(
        mode='train',
        root=cifar_root,
        download_url=download_url,
        transform=compose_transform,
        limit_files=num_samples
)
dataloaders['train_overfit_10samples'] = DataLoader(
        dataset=overfit_dataset,
        batch_size=batch_size,
        shuffle=True,
        drop_last=False,
)

solver = Solver(model, dataloaders['train_overfit_10samples'], dataloaders['val_500files'],
                learning_rate=1e-3, loss_func=loss, optimizer=Adam)

solver.train(epochs=epochs)
```

```
(Epoch 1 / 100) train loss: 2.327229; val loss: 2.327245
(Epoch 2 / 100) train loss: 2.239601; val loss: 2.041127
(Epoch 3 / 100) train loss: 1.863239; val loss: 1.834093
(Epoch 4 / 100) train loss: 1.553708; val loss: 1.673261
(Epoch 5 / 100) train loss: 1.309394; val loss: 1.476275
(Epoch 6 / 100) train loss: 0.870870; val loss: 1.345141
(Epoch 7 / 100) train loss: 0.646340; val loss: 1.238922
(Epoch 8 / 100) train loss: 0.539514; val loss: 1.088024
(Epoch 9 / 100) train loss: 0.427695; val loss: 0.896308
```

```
plt.title('Loss curves')
plt.plot(solver.train_loss_history, '-', label='train')
plt.plot(solver.val_loss_history, '-', label='val')
plt.legend(loc='lower right')
plt.xlabel('Iteration')
plt.show()
```




```
print("Training accuray: %.5f" % (solver.get_dataset_accuracy(dataloaders['train_overfit_10samples'])))
print("Validation accuray: %.5f" % (solver.get_dataset_accuracy(dataloaders['val_500files'])))
```

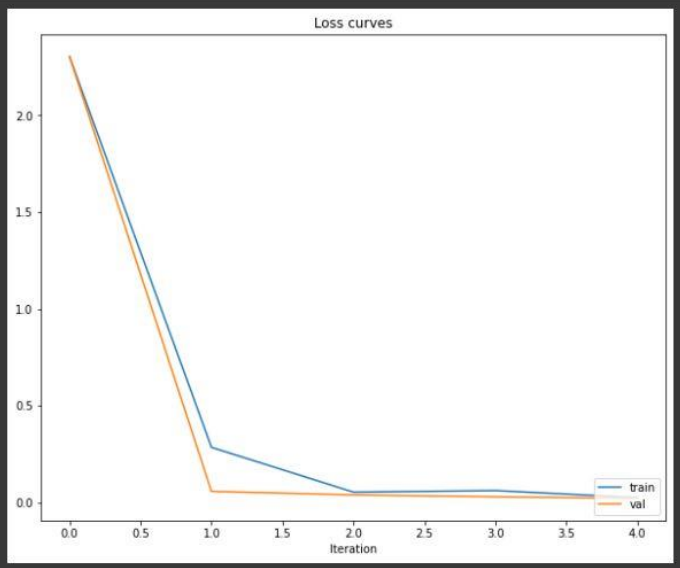
```
Training accuray: 1.00000
Validation accuray: 1.00000
```

測試 network – part III 層數：2 樣本數：1000 reg：0.01

```
from exercise_code.networks import MyOwnNetwork

num_layer = 2
epochs = 5
reg = 0.01

# Make a new data loader with 1000 training samples
num_samples = 1000
overfit_dataset = ImageFolderDataset(
        mode='train',
        root=cifar_root,
        download_url=download_url,
        transform=compose_transform,
        limit_files=num_samples
)
dataloaders['train_small'] = DataLoader(
        dataset=overfit_dataset,
        batch_size=batch_size,
        shuffle=True,
        drop_last=False,
)


# Change here if you want to use the full training set
use_full_training_set = False
if not use_full_training_set:
        train_loader = dataloaders['train_small']
else:
        train_loader = dataloaders['train']


model = ClassificationNet(num_layer=num_layer, reg=reg)
# model = MyOwnNetwork()

loss = CrossEntropyFromLogits()

solver = Solver(model, train_loader, dataloaders['val'],
                                learning_rate=1e-3, loss_func=loss, optimizer=Adam)

solver.train(epochs=epochs)
```

```
(Epoch 1 / 5) train loss: 2.304433; val loss: 2.304426
(Epoch 2 / 5) train loss: 0.285010; val loss: 0.057183
```

```
plt.title('Loss curves')
plt.plot(solver.train_loss_history, '-', label='train')
plt.plot(solver.val_loss_history, '-', label='val')
plt.legend(loc='lower right')
plt.xlabel('Iteration')
plt.show()
```




```
[ ] print("Training accuray: %.5f" % (solver.get_dataset_accuracy(train_loader)))
    print("Validation accuray: %.5f" % (solver.get_dataset_accuracy(dataloaders['val'])))
```

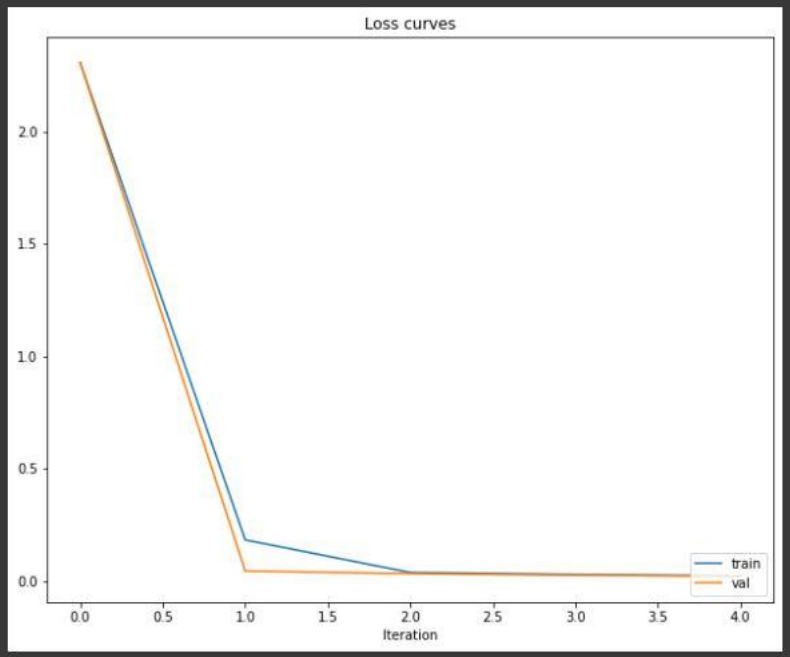
```
Training accuray: 1.00000
Validation accuray: 1.00000
```

測試 network – part III 層數：5 樣本數：1000 reg：0.01

```
from exercise_code.networks import MyOwnNetwork

num_layer = 5
epochs = 5
reg = 0.01

model = ClassificationNet(num_layer=num_layer, reg=reg)
# model = MyOwnNetwork()

# Change here if you want to use the full training set
use_full_training_set = False
if not use_full_training_set:
        train_loader = dataloaders['train_small']
else:
        train_loader = dataloaders['train']

loss = CrossEntropyFromLogits()

solver = Solver(model, train_loader, dataloaders['val'],
                                learning_rate=1e-3, loss_func=loss, optimizer=Adam)

solver.train(epochs=epochs)
```

```
(Epoch 1 / 5) train loss: 2.306118; val loss: 2.306118
(Epoch 2 / 5) train loss: 0.182156; val loss: 0.043832
(Epoch 3 / 5) train loss: 0.037282; val loss: 0.032134
(Epoch 4 / 5) train loss: 0.028330; val loss: 0.024986
(Epoch 5 / 5) train loss: 0.022584; val loss: 0.020575
```

```
plt.title('Loss curves')
plt.plot(solver.train_loss_history, '-', label='train')
plt.plot(solver.val_loss_history, '-', label='val')
plt.legend(loc='lower right')
plt.xlabel('Iteration')
plt.show()
```




```
[ ] print("Training accuray: %.5f" % (solver.get_dataset_accuracy(train_loader)))
    print("Validation accuray: %.5f" % (solver.get_dataset_accuracy(dataloaders['val'])))
```

```
Training accuray: 1.00000
Validation accuray: 1.00000
```

測試 Gird Search

To keep things simple for the beginning, it'll be enough to just focus on the hyperparameters `learning_rate` and `regularization_strength` here, as in the example above.

```
[ ] from exercise_code.networks import MyOwnNetwork

    # Specify the used network
    model_class = ClassificationNet

    from exercise_code import hyperparameter_tuning
    best_model, results = hyperparameter_tuning.grid_search(
            dataloaders['train_small'], dataloaders['val_500files'],
            grid_search_spaces = {
                    "learning_rate": [1e-2, 1e-3, 1e-4],
                    "reg": [1e-4]
            },
            model_class=model_class,
            epochs=10, patience=5)
```

```
Evaluating Config #1 [of 3]:
 {'learning_rate': 0.01, 'reg': 0.0001}
(Epoch 1 / 10) train loss: 2.302146; val loss: 2.302135
(Epoch 2 / 10) train loss: 0.070292; val loss: 0.015778
(Epoch 3 / 10) train loss: 0.006112; val loss: 0.001972
(Epoch 4 / 10) train loss: 0.019049; val loss: 0.203194
(Epoch 5 / 10) train loss: 0.094324; val loss: 0.024246
(Epoch 6 / 10) train loss: 0.011392; val loss: 0.004766
(Epoch 7 / 10) train loss: 0.002781; val loss: 0.001549
(Epoch 8 / 10) train loss: 0.001055; val loss: 0.000724
(Epoch 9 / 10) train loss: 0.022699; val loss: 0.159610
(Epoch 10 / 10) train loss: 0.050168; val loss: 0.011232

Evaluating Config #2 [of 3]:
 {'learning_rate': 0.001, 'reg': 0.0001}
(Epoch 1 / 10) train loss: 2.297308; val loss: 2.297297
(Epoch 2 / 10) train loss: 0.219127; val loss: 0.013786
(Epoch 3 / 10) train loss: 0.008756; val loss: 0.006527
(Epoch 4 / 10) train loss: 0.004881; val loss: 0.004379
```

測試 Random Search

```
from exercise_code.hyperparameter_tuning import random_search
from exercise_code.networks import MyOwnNetwork

# Specify the used network
model_class = ClassificationNet

best_model, results = random_search(
        dataloaders['train_small'], dataloaders['val_500files'],
        random_search_spaces = {
                "learning_rate": ([1e-2, 1e-6], 'log'),
                "reg": ([1e-3, 1e-7], "log"),
                "loss_func": ([CrossEntropyFromLogits()], "item")
        },
        model_class=model_class,
        num_search = 1, epochs=20, patience=5)
```

```
Evaluating Config #1 [of 1]:
 {'learning_rate': 0.0005537359130961477, 'reg': 2.3026500382482102e-05, 'loss_func':
(Epoch 1 / 20) train loss: 2.312254; val loss: 2.312246
(Epoch 2 / 20) train loss: 0.383279; val loss: 0.032240
(Epoch 3 / 20) train loss: 0.019511; val loss: 0.012316
(Epoch 4 / 20) train loss: 0.007835; val loss: 0.007151
(Epoch 5 / 20) train loss: 0.004508; val loss: 0.004805
(Epoch 6 / 20) train loss: 0.003071; val loss: 0.003481
(Epoch 7 / 20) train loss: 0.002283; val loss: 0.002822
(Epoch 8 / 20) train loss: 0.001804; val loss: 0.002295
(Epoch 9 / 20) train loss: 0.001479; val loss: 0.001885
(Epoch 10 / 20) train loss: 0.001253; val loss: 0.001657
(Epoch 11 / 20) train loss: 0.001087; val loss: 0.001447
(Epoch 12 / 20) train loss: 0.000959; val loss: 0.001244
(Epoch 13 / 20) train loss: 0.000859; val loss: 0.001119
(Epoch 14 / 20) train loss: 0.000781; val loss: 0.001027
(Epoch 15 / 20) train loss: 0.000716; val loss: 0.000942
(Epoch 16 / 20) train loss: 0.000662; val loss: 0.000838
(Epoch 17 / 20) train loss: 0.000617; val loss: 0.000766
(Epoch 18 / 20) train loss: 0.000575; val loss: 0.000710
(Epoch 19 / 20) train loss: 0.000542; val loss: 0.000655
(Epoch 20 / 20) train loss: 0.000511; val loss: 0.000620
```

測試自己的 Network

```
from exercise_code.networks import MyOwnNetwork

best_model = ClassificationNet()
#best_model = MyOwnNetwork()

################################################################################
# TODO:                                                                        #
# Implement your own neural network and find suitable hyperparameters    #
# Be sure to edit the MyOwnNetwork class in the following code snippet #
# to upload the correct model!                                                 #
################################################################################

model_class = MyOwnNetwork

best_model, results = random_search(
        dataloaders['train_small'], dataloaders['val_500files'],
        random_search_spaces = {
                "learning_rate": ([1e-2, 1e-6], 'log'),
                "reg": ([1e-3, 1e-7], "log"),
                "loss_func": ([CrossEntropyFromLogits()], "item")
        },
        model_class=model_class,
        num_search = 1, epochs=5, patience=5)

model = MyOwnNetwork(num_layer=num_layer, reg=results[0][0]['reg'])
loss = CrossEntropyFromLogits()
solver = Solver(model, train_loader, dataloaders['val'],
                            learning_rate=results[0][0]['learning_rate'], loss_func=loss, optimizer=Adam)

solver.train(epochs=epochs)

################################################################################
#                                                              END OF YOUR CODE
################################################################################
```

輸出 train loss、val loss

```
Evaluating Config #1 [of 1]:
 {'learning_rate': 2.188855974073908e-06, 'reg': 0.0005370879676704681, 'loss_func': <exercise_code.networks.loss.CrossEntropyFromLogits object at 0x7f779e432150>}
(Epoch 1 / 5) train loss: 2.305668; val loss: 2.305666
(Epoch 2 / 5) train loss: 2.280547; val loss: 2.255058
(Epoch 3 / 5) train loss: 2.229656; val loss: 2.203363
(Epoch 4 / 5) train loss: 2.177519; val loss: 2.150052
(Epoch 5 / 5) train loss: 2.123818; val loss: 2.095085

Search done. Best Val Loss = 2.0950849443403046
Best Config: {'learning_rate': 2.188855974073908e-06, 'reg': 0.0005370879676704681, 'loss_func': <exercise_code.networks.loss.CrossEntropyFromLogits object at 0x7f779e432150>}
(Epoch 1 / 5) train loss: 2.302706; val loss: 2.302706
(Epoch 2 / 5) train loss: 2.277575; val loss: 2.252224
(Epoch 3 / 5) train loss: 2.226878; val loss: 2.201165
(Epoch 4 / 5) train loss: 2.175248; val loss: 2.148889
(Epoch 5 / 5) train loss: 2.122253; val loss: 2.095154
```

測試結果

3.5 Checking the validation accuracy

```
[ ] labels, pred, acc = best_model.get_dataset_prediction(dataloaders['train'])
    print("Train Accuracy: {}%".format(acc*100))
    labels, pred, acc = best_model.get_dataset_prediction(dataloaders['val'])
    print("Validation Accuracy: {}%".format(acc*100))

    Train Accuracy: 100.0%
    Validation Accuracy: 100.0%
```

測試 model 準確度

4. Test your model

When you have finished your hyperparameter tuning and are sure you have your final model that performs well on the validation set (**you should at least get 48% accuracy on the validation set!**), it's time to run your model on the test set.

Important

As you have learned in the lecture, you must only use the test set one single time! So only run the next cell if you are really sure your model works well enough and that you want to submit. Your test set is different from the test set on our server, so results may vary. Nevertheless, you will have a reasonable close approximation about your performance if you only do a final evaluation on the test set.

If you are an external student that can't use our submission webpage: this test performance is your final result and if you surpassed the threshold, you have completed this exercise :). Now, train again to aim for a better number!

```
[ ] # comment this part out to see your model's performance on the test set.
    labels, pred, acc = best_model.get_dataset_prediction(dataloaders['test'])
    print("Test Accuracy: {}%".format(acc*100))
```

```
Test Accuracy: 100.0%
```

儲存檔案

5. Saving your Model

```
[ ] from exercise_code.tests import save_pickle
    save_pickle({"cifar_fcn": best_model}, "cifar_fcn.p")
```

```
[ ] from exercise_code.submit import submit_exercise

    submit_exercise('exercise06')
```

```
relevant folders: ['models']
notebooks files: []
Adding folder models
Zipping successful! Zip is stored under: /content/exercise06.zip
```