

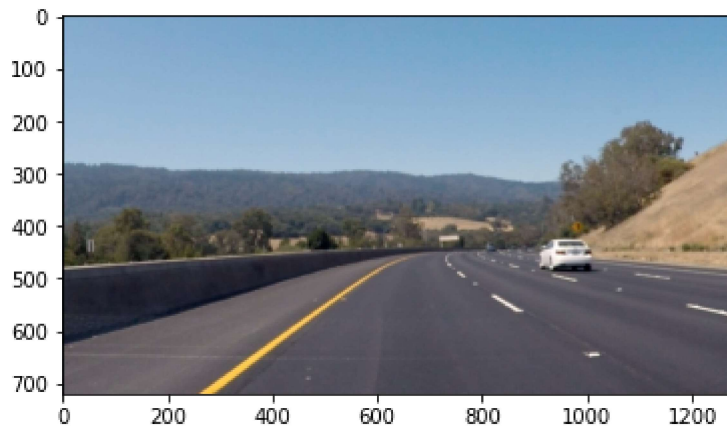
Convolutional Neural Network(CNN):

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

import cv2
```

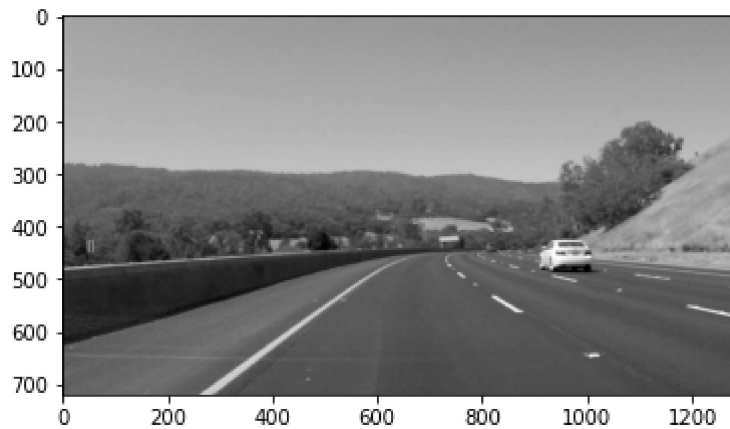
```
In [2]: image = mpimg.imread("curved_lane.jpg")
plt.imshow(image)
```

```
Out[2]: <matplotlib.image.AxesImage at 0x168ea5adfd0>
```



```
In [4]: gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
plt.imshow(gray, cmap = 'gray')
```

```
Out[4]: <matplotlib.image.AxesImage at 0x168eaa92e80>
```

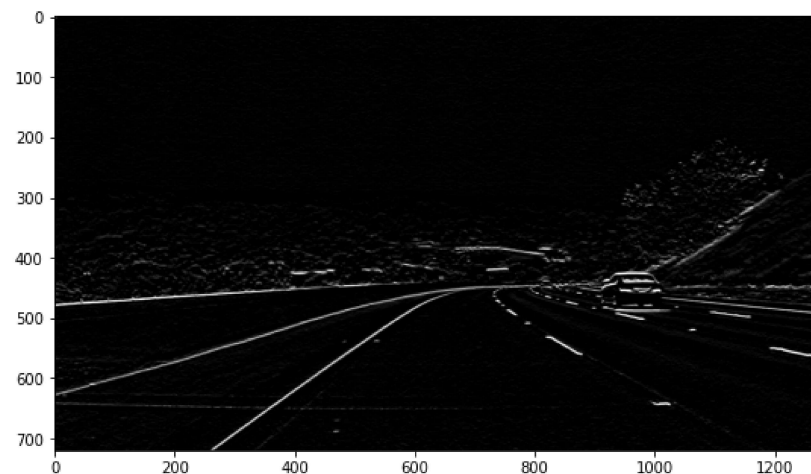
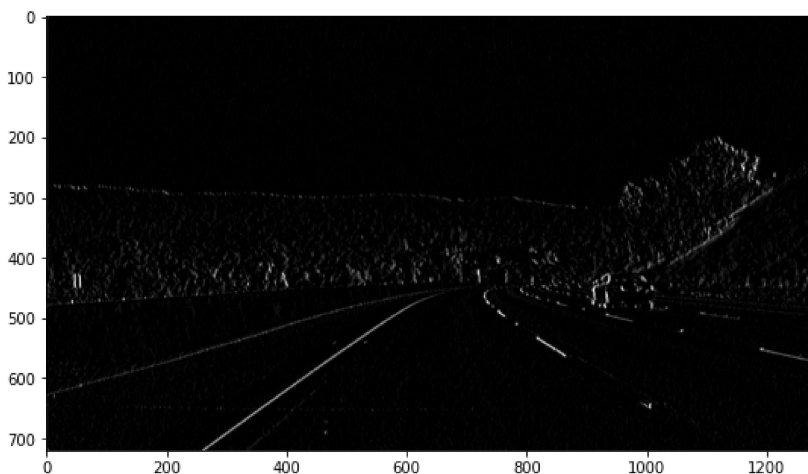


Sobel Operator

```
In [6]: Sobel_x = np.array([[ -1, 0, 1],
                           [ -2, 0, 2],
                           [ -1, 0, 1]])
        Sobel_y = np.array([[ -1, -2, -1],
                           [ 0, 0, 0],
                           [ 1, 2, 1]])
```

```
In [11]: filtered_image = cv2.filter2D(gray, -1, Sobel_x)
        filtered_image2 = cv2.filter2D(gray, -1, Sobel_y)
        fig = plt.figure(figsize=(20, 14))
        fig.add_subplot(1, 2, 1)
        plt.imshow(filtered_image, cmap='gray')
        fig.add_subplot(1, 2, 2)
        plt.imshow(filtered_image2, cmap='gray')
```

```
Out[11]: <matplotlib.image.AxesImage at 0x168ee227340>
```



```
In [19]: import torch
          from torch import nn, optim
          from torchvision import datasets, transforms
          from torch.utils.data.sampler import SubsetRandomSampler
```

```
In [17]: train_on_gpu = torch.cuda.is_available()

          if train_on_gpu:
              print("CUDA is available. Training on GPU.....")
          else:
              print("CUDA is not available. Training on CPU.....")
```

CUDA is available. Training on GPU.....

```
In [21]: train_transform = transforms.Compose([transforms.RandomRotation(10),
                                              transforms.RandomHorizontalFlip(),
                                              transforms.ToTensor(),
                                              transforms.Normalize((0.5,0.5,0.5),
                                                                    (0.5,0.5,0.5))
                                              ])
          test_transform = transforms.Compose([transforms.ToTensor()])
```

```
In [23]: train_data = datasets.CIFAR10("CNN_train/", download = True, train = True, transform = train_transform)
          test_data = datasets.CIFAR10("CNN_test/", download = True, train = False, transform = test_transform)
```

Files already downloaded and verified
Files already downloaded and verified

```
In [24]: trainlen = len(train_data)
val_percent = 0.2
train_indices = list(range(trainlen))
np.random.shuffle(train_indices)
split = int(np.floor(val_percent*trainlen))
valid_idx, train_idx = train_indices[:split], train_indices[split:]
```

```
In [25]: train_sampler = SubsetRandomSampler(train_idx)
valid_sampler = SubsetRandomSampler(valid_idx)
```

```
In [52]: train_loader = torch.utils.data.DataLoader(train_data, batch_size = 32, sampler = train_sampler, num_workers=0)
valid_loader = torch.utils.data.DataLoader(train_data, batch_size = 32, sampler = valid_sampler, num_workers=0)
test_loader = torch.utils.data.DataLoader(test_data, batch_size = 32, shuffle = True, num_workers=0)
```

```
In [27]: classes = ['airplane', 'automobile', 'bird', 'cat', 'deer',
                    'dog', 'frog', 'horse', 'ship', 'truck']
```

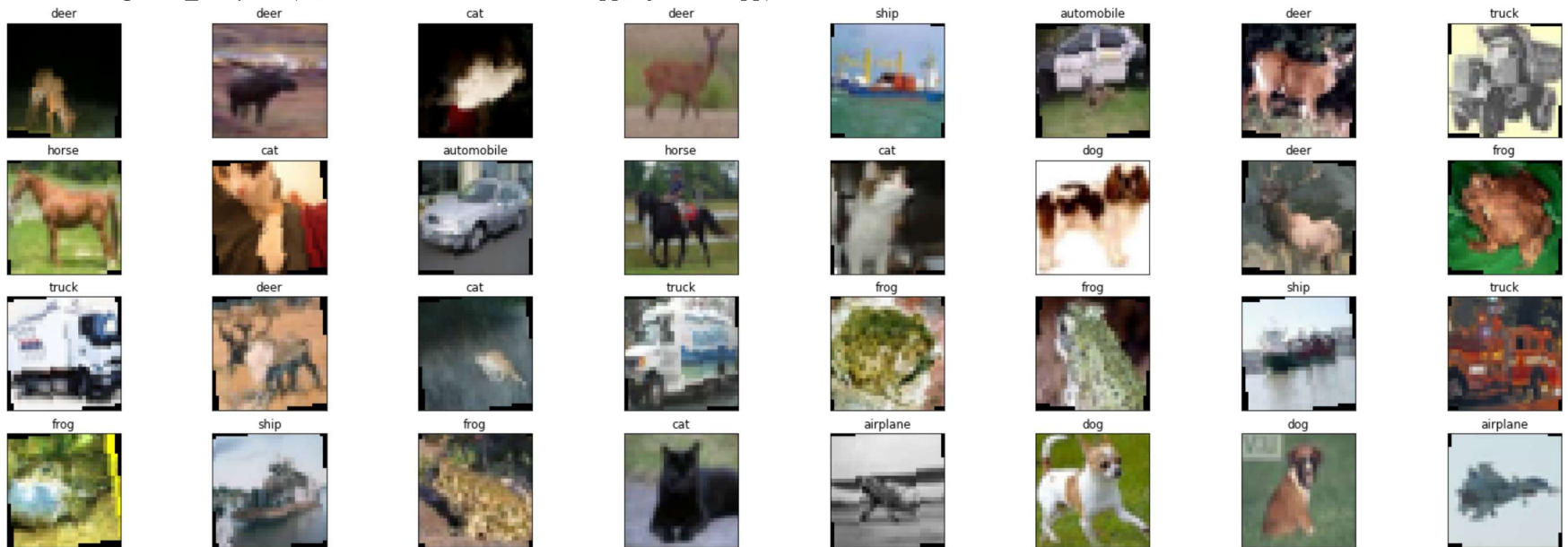
```
In [28]: def imshow(img):
img = img / 2 + 0.5 # unnormalize
plt.imshow(np.transpose(img, (1, 2, 0))) # convert from Tensor image
```

```
In [30]: # obtain one batch of training images
dataiter = iter(train_loader)
images, labels = dataiter.next()
images = images.numpy() # convert images to numpy for display

# plot the images in the batch, along with the corresponding labels
fig = plt.figure(figsize=(30, 10))
# display 20 images
for idx in np.arange(32):
    ax = fig.add_subplot(4, 32/4, idx+1, xticks=[], yticks=[])
    imshow(images[idx])
    ax.set_title(classes[labels[idx]])
```

C:\Users\91720\AppData\Local\Temp\ipykernel_20828\1156741955.py:10: MatplotlibDeprecationWarning: Passing non-integers as three-element position specification is deprecated since 3.3 and will be removed two minor releases later.

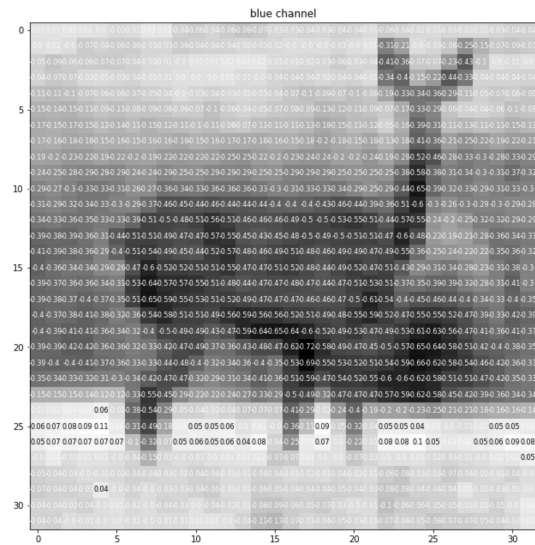
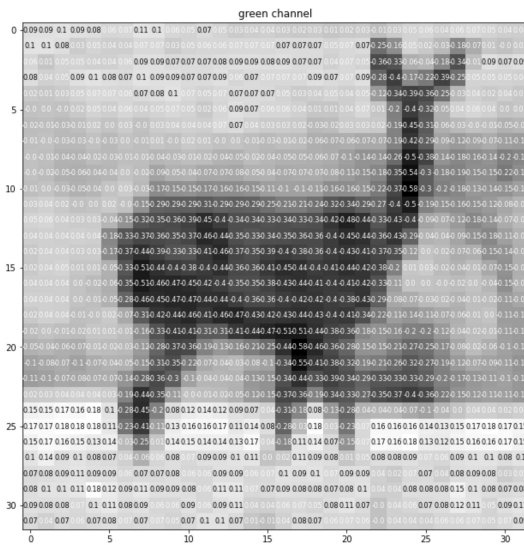
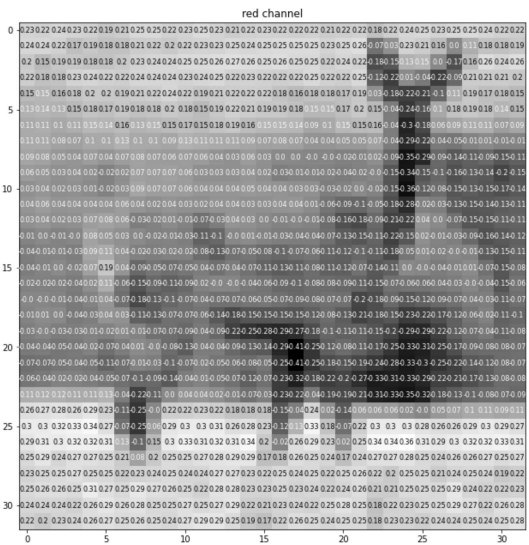
```
ax = fig.add_subplot(4, 32/4, idx+1, xticks=[], yticks=[])
```



In [31]:

```
rgb_img = np.squeeze(images[3])
channels = ['red channel', 'green channel', 'blue channel']

fig = plt.figure(figsize = (36, 36))
for idx in np.arange(rgb_img.shape[0]):
    ax = fig.add_subplot(1, 3, idx + 1)
    img = rgb_img[idx]
    ax.imshow(img, cmap='gray')
    ax.set_title(channels[idx])
    width, height = img.shape
    thresh = img.max()/2.5
    for x in range(width):
        for y in range(height):
            val = round(img[x][y],2) if img[x][y] !=0 else 0
            ax.annotate(str(val), xy=(y,x),
                        horizontalalignment='center',
                        verticalalignment='center', size=8,
                        color='white' if img[x][y]<thresh else 'black')
```



In [39]:

```
import torch.nn.functional as F
```

In [40]:

```
class CNN Model(nn.Module):
```

```
def __init__(self):
    super(CNN_Model, self).__init__()
```

```
self.conv1 = nn.Conv2d(3,16,3,padding = 1)
self.conv2 = nn.Conv2d(16,32,3,padding = 1)
self.conv3 = nn.Conv2d(32,64,3,padding = 1)
```

```
self.pool = nn.MaxPool2d(2,2)
```

```
self.fc1 = nn.Linear(64*4*4,500)
self.fc2 = nn.Linear(500,100)
self.fc3 = nn.Linear(100,10)
```

```
self.dropout = nn.Dropout(0.2)
```

```
def forward(self,x):
    x = F.relu(self.conv1(x))
    x = self.pool(x)
    x = F.relu(self.conv2(x))
    x = self.pool(x)
```

```

        x = F.relu(self.conv3(x))
        x = self.pool(x)

        x = x.view(x.shape[0],-1)

        x = self.dropout(x)

        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
        x = F.softmax(self.fc3(x), dim = 1)

    return x

model = CNN_Model()
if train_on_gpu:
    model.cuda()

criterion = nn.NLLLoss()

optimizer = optim.Adam(model.parameters(),lr = 0.01)

```

In [53]:

```

epoch = 20
train_losses, valid_losses = [], []
step = 0
rn_loss = 0
for i in range(epoch):
    model.train()
    for images,labels in train_loader:
        images,labels = images.cuda(),labels.cuda()

        optimizer.zero_grad()

        output = model.forward(images)
        loss = criterion(output,labels)
        loss.backward()
        optimizer.step()
        rn_loss += loss.item()*images.size(0)

```

```

else:
    model.eval()
    valid_loss = 0
    accuracy = 0
    for inputs, targets in valid_loader:
        inputs, targets = inputs.cuda(), targets.cuda()

        logits = model(inputs)
        val_loss = criterion(logits, targets)

        valid_loss += val_loss.item() * inputs.size(0)

        pred = torch.exp(logits)
        top_p, top_class = pred.topk(1, dim = 1)
        equal = top_class == targets.view(*top_class.shape)
        accuracy += equal.sum().item() * inputs.size(0)

    train_losses.append(rn_loss/len(train_loader.sampler))
    valid_losses.append(accuracy/len(valid_loader.sampler))

    print("Training.....\n",
          "Epoch: {}/{} \n".format(i+1, epoch),
          ".....",
          "Training_loss: {:.3f}".format(rn_loss/len(train_loader.sampler)),
          "Validation_loss: {:.3f}".format(valid_loss/len(valid_loader.sampler)),
          "Accuracy: {:.3f}".format(accuracy/len(valid_loader)))

model.train()

```

```

Training.....
Epoch: 1/20
..... Training_loss: -0.101 Validation_loss: -0.098 Accuracy: 100.038
Training.....
Epoch: 2/20
..... Training_loss: -0.201 Validation_loss: -0.098 Accuracy: 100.192
Training.....
Epoch: 3/20
..... Training_loss: -0.301 Validation_loss: -0.098 Accuracy: 100.089
Training.....
Epoch: 4/20
..... Training_loss: -0.402 Validation_loss: -0.098 Accuracy: 100.089

```

```
Training.....
Epoch: 5/20
..... Training_loss: -0.502 Validation_loss: -0.098 Accuracy: 100.089
Training.....
Epoch: 6/20
..... Training_loss: -0.603 Validation_loss: -0.098 Accuracy: 100.141
Training.....
Epoch: 7/20
..... Training_loss: -0.704 Validation_loss: -0.098 Accuracy: 100.038
Training.....
Epoch: 8/20
..... Training_loss: -0.804 Validation_loss: -0.098 Accuracy: 100.192
Training.....
Epoch: 9/20
..... Training_loss: -0.904 Validation_loss: -0.098 Accuracy: 100.192
Training.....
Epoch: 10/20
..... Training_loss: -1.005 Validation_loss: -0.098 Accuracy: 100.192
Training.....
Epoch: 11/20
..... Training_loss: -1.105 Validation_loss: -0.098 Accuracy: 100.141
Training.....
Epoch: 12/20
..... Training_loss: -1.206 Validation_loss: -0.098 Accuracy: 100.192
Training.....
Epoch: 13/20
..... Training_loss: -1.306 Validation_loss: -0.098 Accuracy: 100.192
Training.....
Epoch: 14/20
..... Training_loss: -1.407 Validation_loss: -0.098 Accuracy: 100.089
Training.....
Epoch: 15/20
..... Training_loss: -1.508 Validation_loss: -0.098 Accuracy: 100.089
Training.....
Epoch: 16/20
..... Training_loss: -1.608 Validation_loss: -0.098 Accuracy: 100.141
Training.....
Epoch: 17/20
..... Training_loss: -1.708 Validation_loss: -0.098 Accuracy: 100.192
Training.....
Epoch: 18/20
..... Training_loss: -1.809 Validation_loss: -0.098 Accuracy: 100.141
Training.....
Epoch: 19/20
..... Training_loss: -1.909 Validation_loss: -0.098 Accuracy: 100.192
```

Training.....

Epoch: 20/20

..... Training_loss: -2.010 Validation_loss: -0.098 Accuracy: 100.192

In [55]:

```
test_loss = 0.0
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
batch_size = 16
model.eval()
# iterate over test data
for batch_idx, (data, target) in enumerate(test_loader):
    # move tensors to GPU if CUDA is available
    if train_on_gpu:
        data, target = data.cuda(), target.cuda()
    # forward pass: compute predicted outputs by passing inputs to the model
    output = model(data)
    # calculate the batch loss
    loss = criterion(output, target)
    # update test loss
    test_loss += loss.item()*data.size(0)
    # convert output probabilities to predicted class
    _, pred = torch.max(output, 1)
    # compare predictions to true label
    correct_tensor = pred.eq(target.data.view_as(pred))
    correct = np.squeeze(correct_tensor.numpy()) if not train_on_gpu else np.squeeze(correct_tensor.cpu().numpy())
    # calculate test accuracy for each object class
    for i in range(batch_size):
        label = target.data[i]
        class_correct[label] += correct[i].item()
        class_total[label] += 1

# average test loss
test_loss = test_loss/len(test_loader.dataset)
print('Test Loss: {:.6f}\n'.format(test_loss))

for i in range(10):
    if class_total[i] > 0:
        print('Test Accuracy of %5s: %2d%% (%2d/%2d)' % (
            classes[i], 100 * class_correct[i] / class_total[i],
            np.sum(class_correct[i]), np.sum(class_total[i])))
    else:
        print('Test Accuracy of %5s: N/A (no training examples)' % (classes[i]))

print('\nTest Accuracy (Overall): %2d%% (%2d/%2d)' % (
```

```
100. * np.sum(class_correct) / np.sum(class_total),  
np.sum(class_correct), np.sum(class_total)))
```

Test Loss: -0.100000

Test Accuracy of airplane: 100% (513/513)

Test Accuracy of automobile: 0% (0/525)

Test Accuracy of bird: 0% (0/511)

Test Accuracy of cat: 0% (0/473)

Test Accuracy of deer: 0% (0/505)

Test Accuracy of dog: 0% (0/481)

Test Accuracy of frog: 0% (0/496)

Test Accuracy of horse: 0% (0/514)

Test Accuracy of ship: 0% (0/512)

Test Accuracy of truck: 0% (0/478)

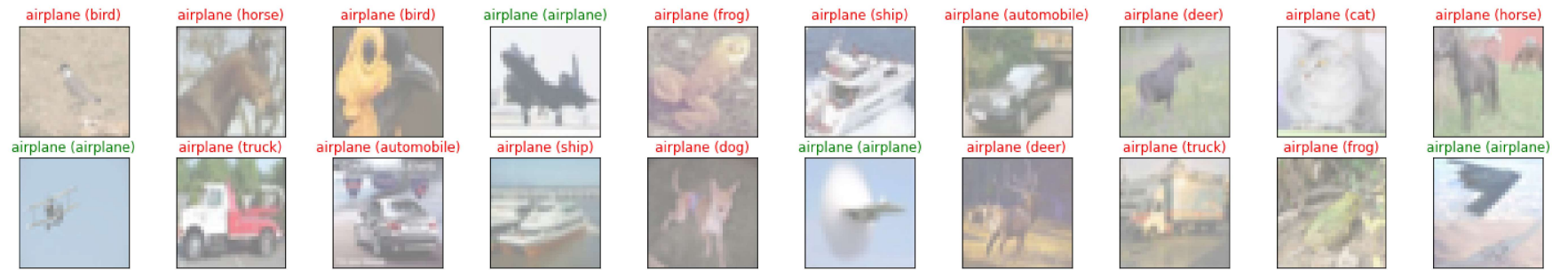
Test Accuracy (Overall): 10% (513/5008)

In [56]:

```
dataiter = iter(test_loader)  
images, labels = dataiter.next()  
images.numpy()  
  
# move model inputs to cuda, if GPU available  
if train_on_gpu:  
    images = images.cuda()  
  
# get sample outputs  
output = model(images)  
# convert output probabilities to predicted class  
_, preds_tensor = torch.max(output, 1)  
preds = np.squeeze(preds_tensor.numpy()) if not train_on_gpu else np.squeeze(preds_tensor.cpu().numpy())  
  
# plot the images in the batch, along with predicted and true labels  
fig = plt.figure(figsize=(25, 4))  
for idx in np.arange(20):  
    ax = fig.add_subplot(2, 20/2, idx+1, xticks=[], yticks=[])  
    imshow(images[idx] if not train_on_gpu else images[idx].cpu())  
    ax.set_title("{} ({}).format(classes[preds[idx]], classes[labels[idx]]),  
                color=("green" if preds[idx]==labels[idx].item() else "red"))
```

C:\Users\91720\AppData\Local\Temp\ipykernel_20828\1293908666.py:18: MatplotlibDeprecationWarning: Passing non-integers as three-element position specification is deprecated since 3.3 and will be removed two minor releases later.

```
ax = fig.add_subplot(2, 20/2, idx+1, xticks=[], yticks=[])
```



In []: