In [ ]:
```
# Reference: Thanks to
# Moon, T. (2018, November 23). Tjmoon0104/pytorch-tiny-imagenet. Retrieved June 14,
    # 2020, from https://github.com/tjmoon0104/pytorch-tiny-imagenet

#Zakka, K. (2017). Train, Validation and Test Split for torchvision Datasets. Retrieved June
    #14, 2020, from https://gist.github.com/kevinzakka/d33bf8d6c7f06a9d8c76d97a7879f5cb

#Xiangjianxiaolu(2018, September 25). Pytorch入门实战：ResNet18图像分类（Cifar10）.
    #Retrieved June 14, 2020, from http://flyrie.top/2018/09/26/Pytorch_ResNet18_Cifar10/
```

setup

In [2]:
```
pip install livelossplot
```

```
Requirement already satisfied: livelossplot in ./.local/lib/python3.7/s
ite-packages (0.5.1)
Requirement already satisfied: bokeh; python_version >= "3.6" in /opt/c
onda/lib/python3.7/site-packages (from livelossplot) (1.3.4)
Requirement already satisfied: matplotlib; python_version >= "3.6" in /
opt/conda/lib/python3.7/site-packages (from livelossplot) (3.1.1)
Requirement already satisfied: ipython in /opt/conda/lib/python3.7/site
-packages (from livelossplot) (7.7.0)
Requirement already satisfied: six>=1.5.2 in /opt/conda/lib/python3.7/s
ite-packages (from bokeh; python_version >= "3.6"->livelossplot) (1.12.
0)
Requirement already satisfied: tornado>=4.3 in /opt/conda/lib/python3.
7/site-packages (from bokeh; python_version >= "3.6"->livelossplot) (6.
0.3)
Requirement already satisfied: python-dateutil>=2.1 in /opt/conda/lib/p
ython3.7/site-packages (from bokeh; python_version >= "3.6"->livelosspl
ot) (2.8.0)
Requirement already satisfied: numpy>=1.7.1 in /opt/conda/lib/python3.
7/site-packages (from bokeh; python_version >= "3.6"->livelossplot) (1.
16.4)
Requirement already satisfied: PyYAML>=3.10 in /opt/conda/lib/python3.
7/site-packages (from bokeh; python_version >= "3.6"->livelossplot) (5.
1.2)
Requirement already satisfied: packaging>=16.8 in /opt/conda/lib/python
3.7/site-packages (from bokeh; python_version >= "3.6"->livelossplot)
(19.0)
Requirement already satisfied: pillow>=4.0 in /opt/conda/lib/python3.7/
site-packages (from bokeh; python_version >= "3.6"->livelossplot) (6.1.
0)
Requirement already satisfied: Jinja2>=2.7 in /opt/conda/lib/python3.7/
site-packages (from bokeh; python_version >= "3.6"->livelossplot) (2.1
0.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1
in /opt/conda/lib/python3.7/site-packages (from matplotlib; python_vers
ion >= "3.6"->livelossplot) (2.4.2)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/pyth
on3.7/site-packages (from matplotlib; python_version >= "3.6"->liveloss
plot) (1.1.0)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.
7/site-packages (from matplotlib; python_version >= "3.6"->livelossplo
t) (0.10.0)
Requirement already satisfied: decorator in /opt/conda/lib/python3.7/si
te-packages (from ipython->livelossplot) (4.4.0)
Requirement already satisfied: pexpect; sys_platform != "win32" in /op
t/conda/lib/python3.7/site-packages (from ipython->livelossplot) (4.7.
0)
Requirement already satisfied: jedi>=0.10 in /opt/conda/lib/python3.7/s
ite-packages (from ipython->livelossplot) (0.14.1)
Requirement already satisfied: backcall in /opt/conda/lib/python3.7/sit
e-packages (from ipython->livelossplot) (0.1.0)
Requirement already satisfied: setuptools>=18.5 in /opt/conda/lib/pytho
n3.7/site-packages (from ipython->livelossplot) (41.0.1)
Requirement already satisfied: prompt-toolkit<2.1.0,>=2.0.0 in /opt/con
da/lib/python3.7/site-packages (from ipython->livelossplot) (2.0.9)
Requirement already satisfied: traitlets>=4.2 in /opt/conda/lib/python
3.7/site-packages (from ipython->livelossplot) (4.3.2)
```

```
Requirement already satisfied: pickleshare in /opt/conda/lib/python3.7/
site-packages (from ipython->livelossplot) (0.7.5)
Requirement already satisfied: pygments in /opt/conda/lib/python3.7/sit
e-packages (from ipython->livelossplot) (2.4.2)
Requirement already satisfied: MarkupSafe>=0.23 in /opt/conda/lib/pytho
n3.7/site-packages (from Jinja2>=2.7->bokeh; python_version >= "3.6"->l
ivelossplot) (1.1.1)
Requirement already satisfied: ptyprocess>=0.5 in /opt/conda/lib/python
3.7/site-packages (from pexpect; sys_platform != "win32"->ipython->live
lossplot) (0.6.0)
Requirement already satisfied: parso>=0.5.0 in /opt/conda/lib/python3.
7/site-packages (from jedi>=0.10->ipython->livelossplot) (0.5.1)
Requirement already satisfied: wcwidth in /opt/conda/lib/python3.7/site
-packages (from prompt-toolkit<2.1.0,>=2.0.0->ipython->livelossplot)
(0.1.7)
Requirement already satisfied: ipython_genutils in /opt/conda/lib/pytho
n3.7/site-packages (from traitlets>=4.2->ipython->livelossplot) (0.2.0)
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
pip install --user opencv-python
```

```
Requirement already satisfied: opencv-python in ./.local/lib/python3.7/
site-packages (4.2.0.34)
Requirement already satisfied: numpy>=1.14.5 in /opt/conda/lib/python3.
7/site-packages (from opencv-python) (1.16.4)
Note: you may need to restart the kernel to use updated packages.
```

In [4]:
```
import torch, os
import torch.nn as nn
import torch.optim as optim
import torchvision.datasets as datasets
import torch.utils.data as data
import torchvision.transforms as transforms
import torchvision.models as models
from train_model import train_model
from test_model import test_model
%matplotlib inline
import torchvision
import torch.utils.model_zoo as model_zoo
from torch.utils.data.sampler import SubsetRandomSampler
from data_loader import get_train_valid_loader
from data_loader import get_test_loader
import torch.nn.functional as F
```

data loader

In [5]:
```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainloader, valloader = get_train_valid_loader('./data', batch_size=10,
augment=0, random_seed=34567,
                                                valid_size=0.02, shuffle
=True, num_workers=2, pin_memory=True)

testset = torchvision.datasets.CIFAR10(root='./data', train=False, downl
oad=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4, shuffle=
False, num_workers=2)

dataloaders = {'train': trainloader, 'val': valloader, 'test': testloade
r}
dataset_sizes = {'train': 49000, 'val': 1000, 'test': len(testset)}

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

```
Files already downloaded and verified
Files already downloaded and verified
Files already downloaded and verified
```

RESNET

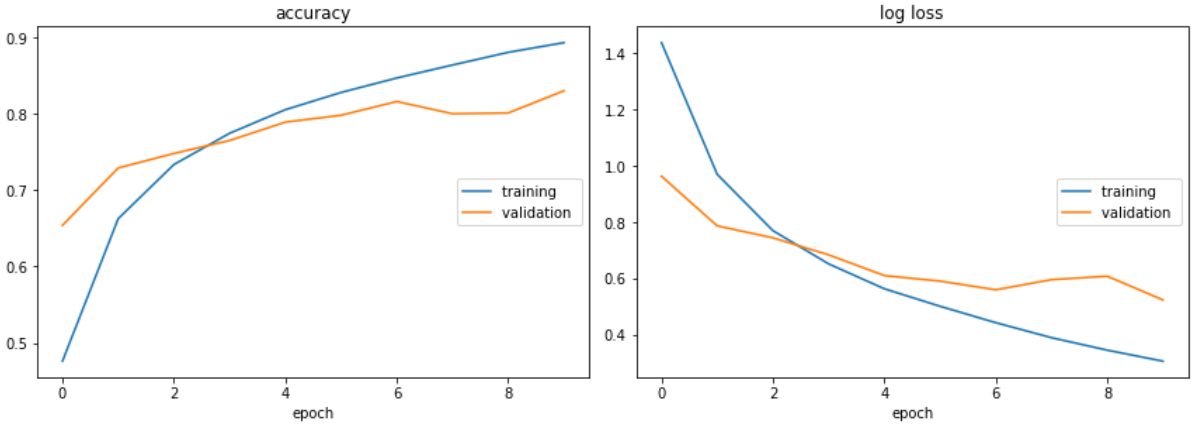
In [6]:
```
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1, **kwargs):
        super().__init__(**kwargs)
        self.residual = nn.Sequential(
            nn.Conv2d(in_channels=in_channels, out_channels=out_channels
, kernel_size=3, stride=stride, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(),
            nn.Conv2d(in_channels=out_channels, out_channels=out_channel
s, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
        )

        self.identity = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.identity = nn.Sequential(
                nn.Conv2d(in_channels=in_channels, out_channels=out_chan
nels, kernel_size=1, stride=stride)
            )

    def forward(self, x):
        out = self.residual(x)
        out += self.identity(x)
        out = F.relu(out)
        return out

class ResNet(nn.Module):
    def __init__(self, num_class=10):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, 3, 1, 1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
        )
        self.layer1 = self._make_layer(64, 64, 2)
        self.layer2 = self._make_layer(64, 128, 2, 2)
        self.layer3 = self._make_layer(128, 256, 2, 2)
        self.layer4 = self._make_layer(256, 512, 2, 2)
        self.avg_pool = nn.AvgPool2d(4)
        self.linear = nn.Linear(512, num_class)

    def _make_layer(self, in_channel, out_channel, bloch_num, stride=1):
        blocks = []
        blocks.append(ResidualBlock(in_channel, out_channel, stride))
        for i in range(1, bloch_num):
            blocks.append(ResidualBlock(out_channel, out_channel))
        return nn.Sequential(*blocks)

    def forward(self, x):
        x = self.conv1(x)
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.avg_pool(x)
```

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

In [15]:
```
def resnet(pretrained=False, **kwargs):
    r"""AlexNet model architecture from the
    `"One weird trick..." <https://arxiv.org/abs/1404.5997>`_ paper.
    Args:
        pretrained (bool): If True, returns a model pre-trained on Image
Net
    """
    model = ResNet(**kwargs)
#     if pretrained:
#         model.load_state_dict(model_zoo.load_url(model_urls['alexne
t']))
#     model.classifier[1] = nn.Linear(256 * 2 * 2, 4096)
#     model.classifier[6] = nn.Linear(4096, 10)
    return model
```

Train

In [10]:
```
#Load Resnet18
model_ft = resnet()
#Finetune Final few layers to adjust for tiny imagenet input
# model_ft.avgpool = nn.AdaptiveAvgPool2d(1)
# model_ft.fc.out_features = 10
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft = model_ft.to(device)
#Loss Function
criterion = nn.CrossEntropyLoss()
# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

#Train
train_model("64_no_pre_kelly", model_ft, dataloaders, dataset_sizes, cri
terion, optimizer_ft, num_epochs=10)
```




```
accuracy
        training                (min:    0.476, max:    0.893, cur:
0.893)
        validation              (min:    0.654, max:    0.830, cur:
0.830)
log loss
        training                (min:    0.306, max:    1.437, cur:
0.306)
        validation              (min:    0.523, max:    0.962, cur:
0.523)
Train Loss: 0.3057 Acc: 0.8930
Val Loss: 0.5232 Acc: 0.8300

Training complete in 34m 23s
Best Validation Accuracy: 0.8300000000000001, Epoch: 10
```

Test

In [25]:
```
#Test Resnet18
model_ft = resnet()
#Finetune Final few layers to adjust for tiny imagenet input
# model_ft.avgpool = nn.AdaptiveAvgPool2d(1)
# model_ft.fc.out_features = 10
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft.load_state_dict(torch.load('models/64_no_pre_kelly/model_10_epoch.pt')) #TODO
model_ft = model_ft.to(device)

#Loss Function
criterion = nn.CrossEntropyLoss()
# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

#Test
test_model(model_ft, dataloaders, dataset_sizes, criterion, optimizer_ft)
```

```
Iteration: 2500/2500, Loss: 5.67810583114624....
Test Loss: 0.9124 Acc: 0.7029

Test complete in 0m 27s
```

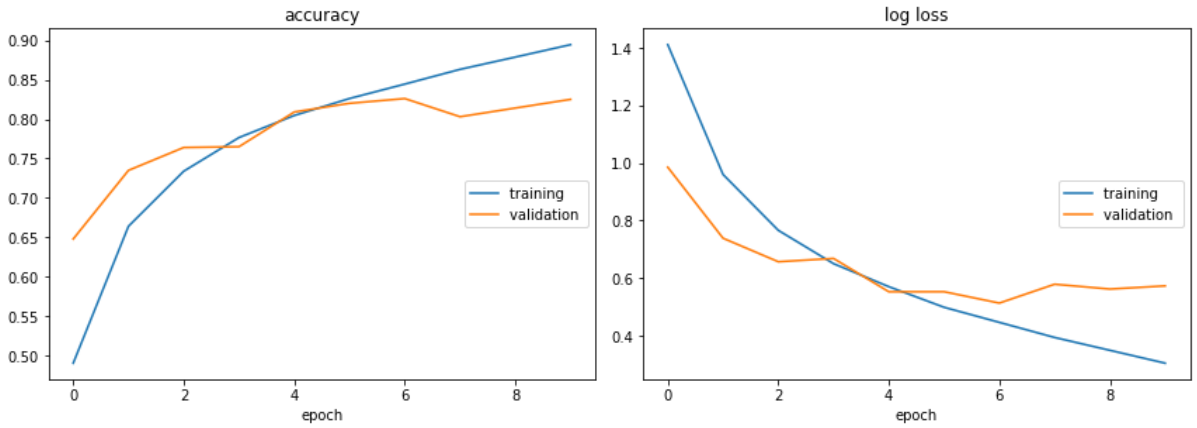
RESNET -activation: leakyrelu

In [30]:

```
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1, **kwargs):
        super().__init__(**kwargs)
        self.residual = nn.Sequential(
            nn.Conv2d(in_channels=in_channels, out_channels=out_channels
, kernel_size=3, stride=stride, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.LeakyReLU(inplace = True),
            nn.Conv2d(in_channels=out_channels, out_channels=out_channel
s, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
        )

        self.identity = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.identity = nn.Sequential(
                nn.Conv2d(in_channels=in_channels, out_channels=out_chan
nels, kernel_size=1, stride=stride)
            )

    def forward(self, x):
        out = self.residual(x)
        out += self.identity(x)
        out = F.relu(out)
        return out

class ResNet(nn.Module):
    def __init__(self, num_class=10):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, 3, 1, 1),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(inplace = True),
        )
        self.layer1 = self._make_layer(64, 64, 2)
        self.layer2 = self._make_layer(64, 128, 2, 2)
        self.layer3 = self._make_layer(128, 256, 2, 2)
        self.layer4 = self._make_layer(256, 512, 2, 2)
        self.avg_pool = nn.AvgPool2d(4)
        self.linear = nn.Linear(512, num_class)

    def _make_layer(self, in_channel, out_channel, bloch_num, stride=1):
        blocks = []
        blocks.append(ResidualBlock(in_channel, out_channel, stride))
        for i in range(1, bloch_num):
            blocks.append(ResidualBlock(out_channel, out_channel))
        return nn.Sequential(*blocks)

    def forward(self, x):
        x = self.conv1(x)
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.avg_pool(x)
```

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

In [31]:
```
#Load Resnet18
model_ft = resnet()
#Finetune Final few layers to adjust for tiny imagenet input
# model_ft.avgpool = nn.AdaptiveAvgPool2d(1)
# model_ft.fc.out_features = 10
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft = model_ft.to(device)
#Loss Function
criterion = nn.CrossEntropyLoss()
# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

#Train
train_model("64_no_pre_kelly_leaky", model_ft, dataloaders, dataset_size
s, criterion, optimizer_ft, num_epochs=10)
```




```
accuracy
        training                (min:    0.491, max:    0.894, cur:
0.894)
        validation              (min:    0.648, max:    0.826, cur:
0.825)
log loss
        training                (min:    0.304, max:    1.411, cur:
0.304)
        validation              (min:    0.513, max:    0.985, cur:
0.573)
Train Loss: 0.3040 Acc: 0.8943
Val Loss: 0.5725 Acc: 0.8250

Training complete in 33m 28s
Best Validation Accuracy: 0.8260000000000001, Epoch: 7
```

In [32]:
```
#Test Resnet18
model_ft = resnet()
#Finetune Final few layers to adjust for tiny imagenet input
# model_ft.avgpool = nn.AdaptiveAvgPool2d(1)
# model_ft.fc.out_features = 10
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft.load_state_dict(torch.load('models/64_no_pre_kelly_leaky/model_
7_epoch.pt')) #TODO
model_ft = model_ft.to(device)

#Loss Function
criterion = nn.CrossEntropyLoss()
# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

#Test
test_model(model_ft, dataloaders, dataset_sizes, criterion, optimizer_ft
)
```

```
Iteration: 2500/2500, Loss: 3.1883418560028076..
Test Loss: 0.8839 Acc: 0.7106

Test complete in 0m 27s
```

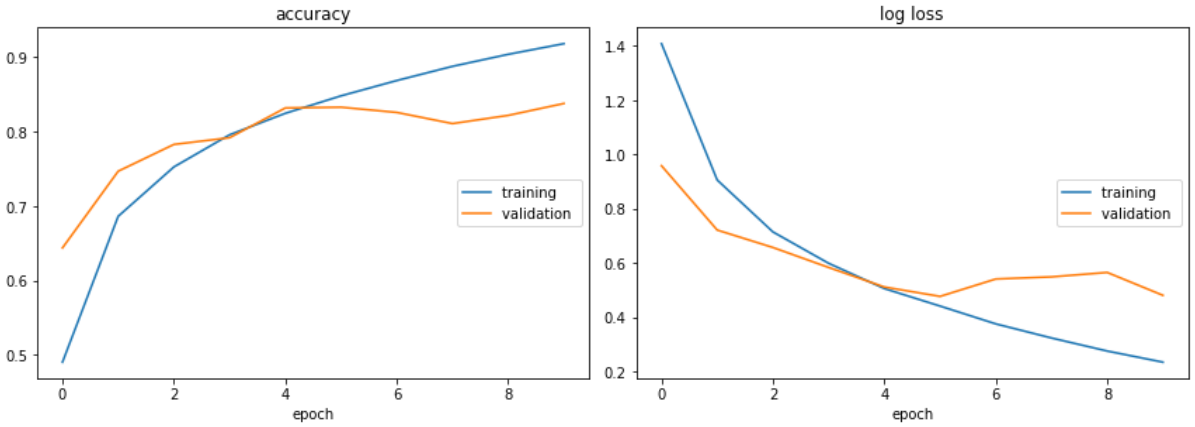
RESNET - optimizer： Adam

In [7]:
```
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1, **kwargs):
        super().__init__(**kwargs)
        self.residual = nn.Sequential(
            nn.Conv2d(in_channels=in_channels, out_channels=out_channels
, kernel_size=3, stride=stride, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(),
            nn.Conv2d(in_channels=out_channels, out_channels=out_channel
s, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
        )

        self.identity = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.identity = nn.Sequential(
                nn.Conv2d(in_channels=in_channels, out_channels=out_chan
nels, kernel_size=1, stride=stride)
            )

    def forward(self, x):
        out = self.residual(x)
        out += self.identity(x)
        out = F.relu(out)
        return out

class ResNet(nn.Module):
    def __init__(self, num_class=10):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, 3, 1, 1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
        )
        self.layer1 = self._make_layer(64, 64, 2)
        self.layer2 = self._make_layer(64, 128, 2, 2)
        self.layer3 = self._make_layer(128, 256, 2, 2)
        self.layer4 = self._make_layer(256, 512, 2, 2)
        self.avg_pool = nn.AvgPool2d(4)
        self.linear = nn.Linear(512, num_class)

    def _make_layer(self, in_channel, out_channel, bloch_num, stride=1):
        blocks = []
        blocks.append(ResidualBlock(in_channel, out_channel, stride))
        for i in range(1, bloch_num):
            blocks.append(ResidualBlock(out_channel, out_channel))
        return nn.Sequential(*blocks)

    def forward(self, x):
        x = self.conv1(x)
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.avg_pool(x)
```

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

In [34]:
```
#Load AlexNet
model_ft = resnet()

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft = model_ft.to(device)

#Loss Function
criterion = nn.CrossEntropyLoss()
# Observe that all parameters are being optimized
optimizer_ft = optim.Adam(model_ft.parameters(), lr=0.001)
```

In [35]:
```
train_model("resnet_adam",model_ft, dataloaders, dataset_sizes, criterio
n, optimizer_ft, num_epochs=10)
```




```
accuracy
        training                (min:    0.490, max:    0.918, cur:
0.918)
        validation              (min:    0.644, max:    0.838, cur:
0.838)
log loss
        training                (min:    0.235, max:    1.408, cur:
0.235)
        validation              (min:    0.477, max:    0.958, cur:
0.481)
Train Loss: 0.2347 Acc: 0.9184
Val Loss: 0.4809 Acc: 0.8380

Training complete in 53m 47s
Best Validation Accuracy: 0.838, Epoch: 10
```

In [36]:
```
#Test Resnet18
model_ft = resnet()
#Finetune Final few layers to adjust for tiny imagenet input
# model_ft.avgpool = nn.AdaptiveAvgPool2d(1)
# model_ft.fc.out_features = 10
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft.load_state_dict(torch.load('models/resnet_adam/model_10_epoch.pt')) #TODO
model_ft = model_ft.to(device)

#Loss Function
criterion = nn.CrossEntropyLoss()
# Observe that all parameters are being optimized
optimizer_ft = optim.Adam(model_ft.parameters(), lr=0.001)

#Test
test_model(model_ft, dataloaders, dataset_sizes, criterion, optimizer_ft)
```

```
Iteration: 2500/2500, Loss: 7.171634674072266.....
Test Loss: 0.7421 Acc: 0.7644

Test complete in 0m 28s
```

RESNET - fine tune

In [8]:
```
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1, **kwargs):
        super().__init__(**kwargs)
        self.residual = nn.Sequential(
            nn.Conv2d(in_channels=in_channels, out_channels=out_channels
, kernel_size=3, stride=stride, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(),
            nn.Conv2d(in_channels=out_channels, out_channels=out_channel
s, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
        )

        self.identity = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.identity = nn.Sequential(
                nn.Conv2d(in_channels=in_channels, out_channels=out_chan
nels, kernel_size=1, stride=stride)
            )

    def forward(self, x):
        out = self.residual(x)
        out += self.identity(x)
        out = F.relu(out)
        return out

class ResNet(nn.Module):
    def __init__(self, num_class=10):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, 3, 1, 1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
        )
        self.layer1 = self._make_layer(64, 64, 2)
        self.layer2 = self._make_layer(64, 128, 2, 2)
        self.layer3 = self._make_layer(128, 256, 2, 2)
        self.layer4 = self._make_layer(256, 512, 2, 2)
        self.avg_pool = nn.AvgPool2d(4)
        self.linear = nn.Linear(512, num_class)

    def _make_layer(self, in_channel, out_channel, bloch_num, stride=1):
        blocks = []
        blocks.append(ResidualBlock(in_channel, out_channel, stride))
        for i in range(1, bloch_num):
            blocks.append(ResidualBlock(out_channel, out_channel))
        return nn.Sequential(*blocks)

    def forward(self, x):
        x = self.conv1(x)
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.avg_pool(x)
```

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

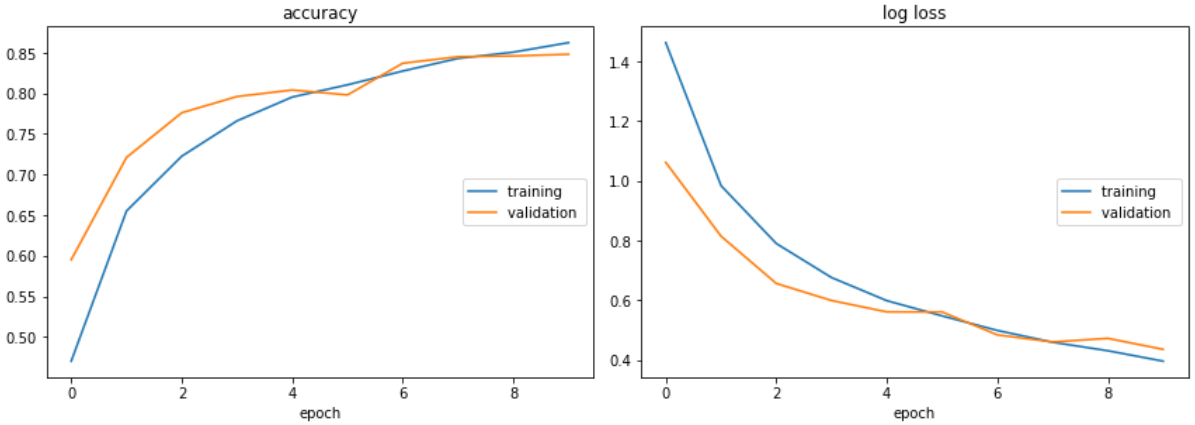
aug1

In [9]:

```
trainloader, valloader = get_train_valid_loader('./data', batch_size=10,
augment=1, random_seed=34567,
                                                valid_size=0.02, shuffle
=True, num_workers=2, pin_memory=True)

dataloaders = {'train': trainloader, 'val': valloader, 'test': testloade
r}
dataset_sizes = {'train': 49000, 'val': 1000, 'test': len(testset)}
```

```
Files already downloaded and verified
Files already downloaded and verified
```

In [39]:
```
#Load Resnet18
model_ft = resnet()
#Finetune Final few layers to adjust for tiny imagenet input
# model_ft.avgpool = nn.AdaptiveAvgPool2d(1)
# model_ft.fc.out_features = 10
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft = model_ft.to(device)
#Loss Function
criterion = nn.CrossEntropyLoss()
# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

#Train
train_model("resnet_aug1", model_ft, dataloaders, dataset_sizes, criteri
on, optimizer_ft, num_epochs=10)
```




```
accuracy
        training                (min:    0.470, max:    0.862, cur:
0.862)
        validation              (min:    0.595, max:    0.848, cur:
0.848)
log loss
        training                (min:    0.396, max:    1.462, cur:
0.396)
        validation              (min:    0.436, max:    1.061, cur:
0.436)
Train Loss: 0.3962 Acc: 0.8623
Val Loss: 0.4358 Acc: 0.8480

Training complete in 33m 52s
Best Validation Accuracy: 0.848, Epoch: 10
```

In [40]:
```
#Test Resnet18
model_ft = resnet()
#Finetune Final few layers to adjust for tiny imagenet input
# model_ft.avgpool = nn.AdaptiveAvgPool2d(1)
# model_ft.fc.out_features = 10
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft.load_state_dict(torch.load('models/resnet_aug1/model_10_epoch.pt')) #TODO
model_ft = model_ft.to(device)

#Loss Function
criterion = nn.CrossEntropyLoss()
# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

#Test
test_model(model_ft, dataloaders, dataset_sizes, criterion, optimizer_ft)
```

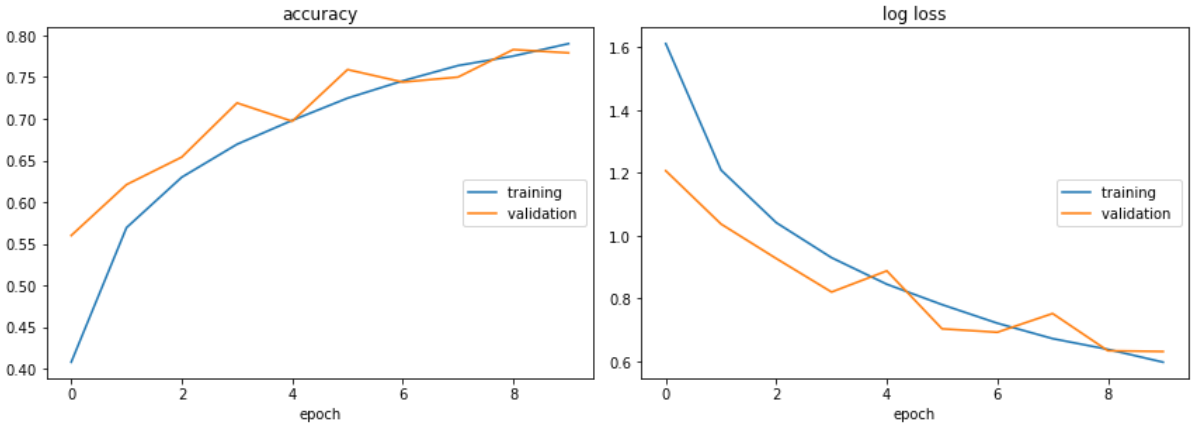
```
Iteration: 2500/2500, Loss: 3.2925541400909424...
Test Loss: 0.7696 Acc: 0.7391

Test complete in 0m 29s
```

aug2

In [10]:
```
trainloader, valloader = get_train_valid_loader('./data', batch_size=10, augment=2, random_seed=34567,
                                                valid_size=0.02, shuffle=True, num_workers=2, pin_memory=True)

dataloaders = {'train': trainloader, 'val': valloader, 'test': testloader}
dataset_sizes = {'train': 49000, 'val': 1000, 'test': len(testset)}
```

```
Files already downloaded and verified
Files already downloaded and verified
```

In [42]:
```
#Load Resnet18
model_ft = resnet()
#Finetune Final few layers to adjust for tiny imagenet input
# model_ft.avgpool = nn.AdaptiveAvgPool2d(1)
# model_ft.fc.out_features = 10
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft = model_ft.to(device)
#Loss Function
criterion = nn.CrossEntropyLoss()
# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

#Train
train_model("resnet_aug2", model_ft, dataloaders, dataset_sizes, criteri
on, optimizer_ft, num_epochs=10)
```




```
accuracy
        training                (min:    0.408, max:    0.790, cur:
0.790)
        validation              (min:    0.560, max:    0.783, cur:
0.779)
log loss
        training                (min:    0.597, max:    1.612, cur:
0.597)
        validation              (min:    0.631, max:    1.207, cur:
0.631)
Train Loss: 0.5975 Acc: 0.7901
Val Loss: 0.6315 Acc: 0.7790

Training complete in 34m 33s
Best Validation Accuracy: 0.783, Epoch: 9
```

In [11]:
```
#Test Resnet18
model_ft = resnet()
#Finetune Final few layers to adjust for tiny imagenet input
# model_ft.avgpool = nn.AdaptiveAvgPool2d(1)
# model_ft.fc.out_features = 10
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft.load_state_dict(torch.load('models/resnet_aug2/model_9_epoch.p
t')) #TODO
model_ft = model_ft.to(device)

#Loss Function
criterion = nn.CrossEntropyLoss()
# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

#Test
test_model(model_ft, dataloaders, dataset_sizes, criterion, optimizer_ft
)
```

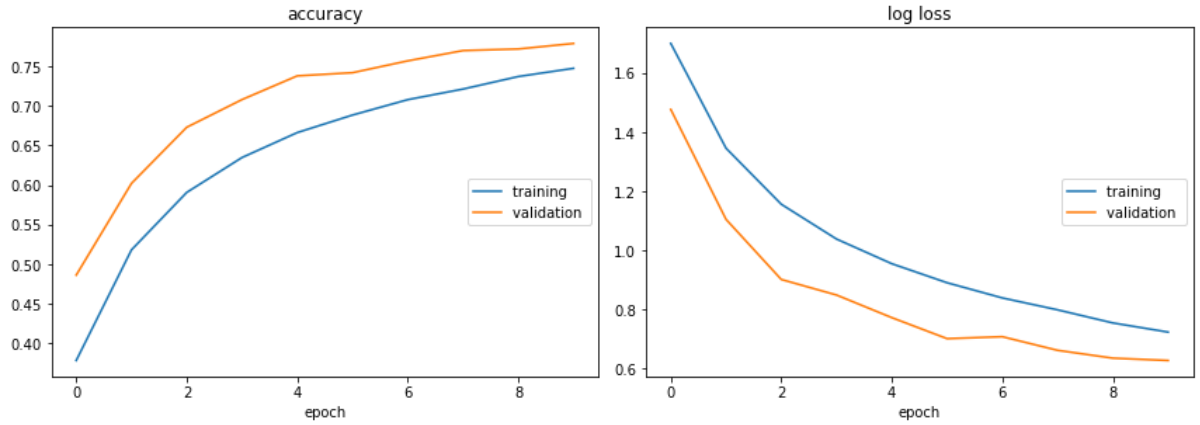
```
Iteration: 2500/2500, Loss: 2.306704521179199....
Test Loss: 0.9839 Acc: 0.6721

Test complete in 0m 34s
```

aug3

In [12]:
```
trainloader, valloader = get_train_valid_loader('./data', batch_size=10,
augment=3, random_seed=34567,
                                                valid_size=0.02, shuffle
=True, num_workers=2, pin_memory=True)

dataloaders = {'train': trainloader, 'val': valloader, 'test': testloade
r}
dataset_sizes = {'train': 49000, 'val': 1000, 'test': len(testset)}
```

```
Files already downloaded and verified
Files already downloaded and verified
```

In [47]:
```
#Load Resnet18
model_ft = resnet()
#Finetune Final few layers to adjust for tiny imagenet input
# model_ft.avgpool = nn.AdaptiveAvgPool2d(1)
# model_ft.fc.out_features = 10
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft = model_ft.to(device)
#Loss Function
criterion = nn.CrossEntropyLoss()
# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

#Train
train_model("resnet_aug3", model_ft, dataloaders, dataset_sizes, criteri
on, optimizer_ft, num_epochs=10)
```




```
accuracy
        training                (min:    0.378, max:    0.748, cur:
0.748)
        validation              (min:    0.486, max:    0.779, cur:
0.779)
log loss
        training                (min:    0.724, max:    1.700, cur:
0.724)
        validation              (min:    0.628, max:    1.476, cur:
0.628)
Train Loss: 0.7242 Acc: 0.7476
Val Loss: 0.6283 Acc: 0.7790

Training complete in 34m 47s
Best Validation Accuracy: 0.779, Epoch: 10
```

In [48]:
```
#Test Resnet18
model_ft = resnet()
#Finetune Final few layers to adjust for tiny imagenet input
# model_ft.avgpool = nn.AdaptiveAvgPool2d(1)
# model_ft.fc.out_features = 10
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft.load_state_dict(torch.load('models/resnet_aug3/model_10_epoch.pt')) #TODO
model_ft = model_ft.to(device)

#Loss Function
criterion = nn.CrossEntropyLoss()
# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

#Test
test_model(model_ft, dataloaders, dataset_sizes, criterion, optimizer_ft)
```

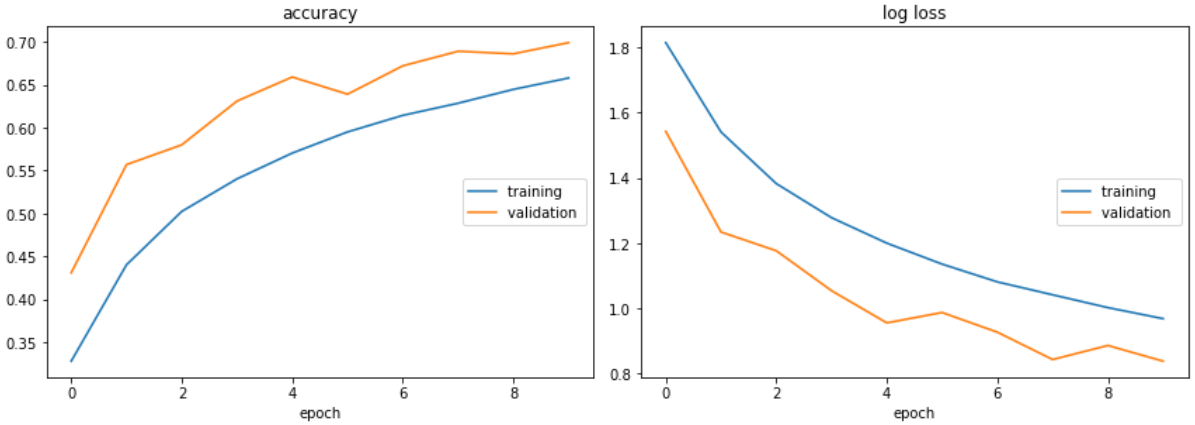
```
Iteration: 2500/2500, Loss: 4.481856822967529...
Test Loss: 1.5099 Acc: 0.5551

Test complete in 0m 28s
```

aug4

In [13]:
```
trainloader, valloader = get_train_valid_loader('./data', batch_size=10, augment=4, random_seed=34567,
                                                valid_size=0.02, shuffle=True, num_workers=2, pin_memory=True)

dataloaders = {'train': trainloader, 'val': valloader, 'test': testloader}
dataset_sizes = {'train': 49000, 'val': 1000, 'test': len(testset)}
```

```
Files already downloaded and verified
Files already downloaded and verified
```

In [50]:
```
#Load Resnet18
model_ft = resnet()
#Finetune Final few layers to adjust for tiny imagenet input
# model_ft.avgpool = nn.AdaptiveAvgPool2d(1)
# model_ft.fc.out_features = 10
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft = model_ft.to(device)
#Loss Function
criterion = nn.CrossEntropyLoss()
# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

#Train
train_model("resnet_aug4", model_ft, dataloaders, dataset_sizes, criteri
on, optimizer_ft, num_epochs=10)
```




```
accuracy
        training                (min:    0.328, max:    0.658, cur:
0.658)
        validation              (min:    0.431, max:    0.699, cur:
0.699)
log loss
        training                (min:    0.968, max:    1.815, cur:
0.968)
        validation              (min:    0.838, max:    1.543, cur:
0.838)
Train Loss: 0.9682 Acc: 0.6578
Val Loss: 0.8378 Acc: 0.6990

Training complete in 33m 38s
Best Validation Accuracy: 0.6990000000000001, Epoch: 10
```

In [51]:

```
#Test Resnet18
model_ft = resnet()
#Finetune Final few layers to adjust for tiny imagenet input
# model_ft.avgpool = nn.AdaptiveAvgPool2d(1)
# model_ft.fc.out_features = 10
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft.load_state_dict(torch.load('models/resnet_aug4/model_10_epoch.pt')) #TODO
model_ft = model_ft.to(device)

#Loss Function
criterion = nn.CrossEntropyLoss()
# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

#Test
test_model(model_ft, dataloaders, dataset_sizes, criterion, optimizer_ft)
```

```
Iteration: 2500/2500, Loss: 4.068412780761719...
Test Loss: 1.6114 Acc: 0.4739

Test complete in 0m 29s
```