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

#Zakka, K. (2017). Train, Validation and Test Split for torchvision Data
sets. Retrieved June
    #14, 2020, from https://gist.github.com/kevinzakka/d33bf8d6c7f06a9d8
c76d97a7879f5cb

#Luo, Z. (2018, September 21). Icpm/pytorch-cifar10. Retrieved June 14,
 2020, from
    #https://github.com/icpm/pytorch-cifar10/tree/master/models
```

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: 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: tornado>=4.3 in /opt/conda/lib/python3.
7/site-packages (from bokeh; python_version >= "3.6"->livelossplot) (6.
0.3)
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: 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: packaging>=16.8 in /opt/conda/lib/python
3.7/site-packages (from bokeh; python_version >= "3.6"->livelossplot)
(19.0)
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: pillow>=4.0 in /opt/conda/lib/python3.7/
site-packages (from bokeh; python_version >= "3.6"->livelossplot) (6.1.
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: 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: 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: pickleshare in /opt/conda/lib/python3.7/
site-packages (from ipython->livelossplot) (0.7.5)
Requirement already satisfied: traitlets>=4.2 in /opt/conda/lib/python
3.7/site-packages (from ipython->livelossplot) (4.3.2)
Requirement already satisfied: backcall in /opt/conda/lib/python3.7/sit
e-packages (from ipython->livelossplot) (0.1.0)
Requirement already satisfied: decorator in /opt/conda/lib/python3.7/si
te-packages (from ipython->livelossplot) (4.4.0)
Requirement already satisfied: setuptools>=18.5 in /opt/conda/lib/pytho
n3.7/site-packages (from ipython->livelossplot) (41.0.1)
Requirement already satisfied: 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: 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: 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)
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)
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
```

Load data

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=4,
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
```

In [6]:
```
print(dataloaders)
print(dataset_sizes)
```

```
{'train': <torch.utils.data.dataloader.DataLoader object at 0x7fccb095a
c50>, 'val': <torch.utils.data.dataloader.DataLoader object at 0x7fccb0
95aba8>, 'test': <torch.utils.data.dataloader.DataLoader object at 0x7f
ccb095ae48>}
{'train': 49000, 'val': 1000, 'test': 10000}
```

Alexnet

In [7]:
```
__all__ = ['AlexNet', 'alexnet']


model_urls = {
    'alexnet': 'https://download.pytorch.org/models/alexnet-owt-4df8aa7
1.pth',
}
```

Alexnet- Relu

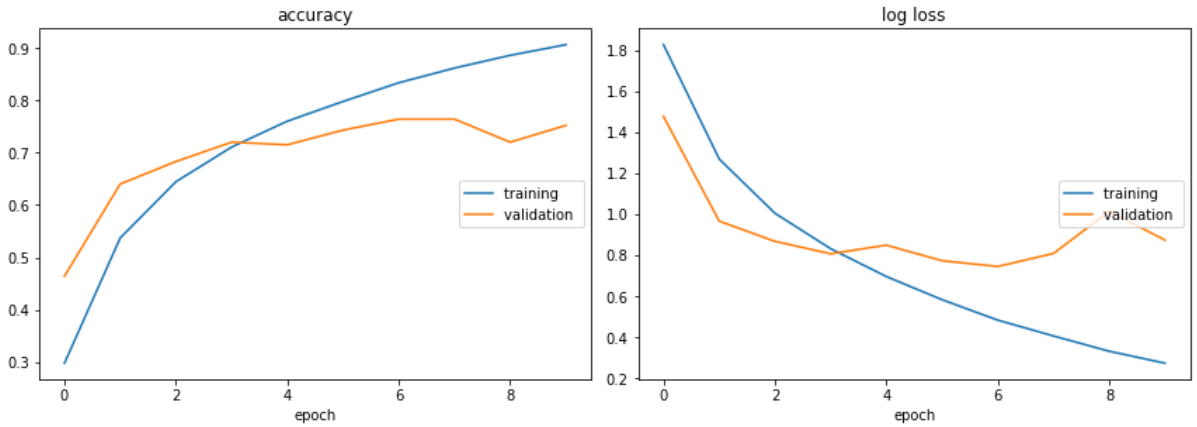
In [8]:
```
class AlexNet(nn.Module):

    def __init__(self, num_classes=10):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=2, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(64, 192, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2),
        )
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 2 * 2, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), 256 * 2 * 2)
        x = self.classifier(x)
        return x
```

In [9]:
```
def alexnet(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 = AlexNet(**kwargs)
    if pretrained:
        model.load_state_dict(model_zoo.load_url(model_urls['alexnet']))
    model.classifier[1] = nn.Linear(256 * 2 * 2, 4096)
    model.classifier[6] = nn.Linear(4096, 10)
    return model
```

optimizer: SGD

In [10]:
```
#Load AlexNet
model_ft = alexnet()

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)
```

In [11]:
```
train_model("Alex_sgd",model_ft, dataloaders, dataset_sizes, criterion,
optimizer_ft, num_epochs=10)
```




```
accuracy
        training                (min:    0.298, max:    0.906, cur:
0.906)
        validation              (min:    0.464, max:    0.764, cur:
0.752)
log loss
        training                (min:    0.271, max:    1.826, cur:
0.271)
        validation              (min:    0.744, max:    1.476, cur:
0.872)
Train Loss: 0.2715 Acc: 0.9062
Val Loss: 0.8724 Acc: 0.7520

Training complete in 23m 22s
Best Validation Accuracy: 0.764, Epoch: 7
```

In [13]:
```
#Load AlexNet
model_ft = alexnet()

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft.load_state_dict(torch.load('models/Alex_sgd/model_7_epoch.pt'))
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: 17.335664749145508..
Test Loss: 1.9669 Acc: 0.4743

Test complete in 0m 12s
```

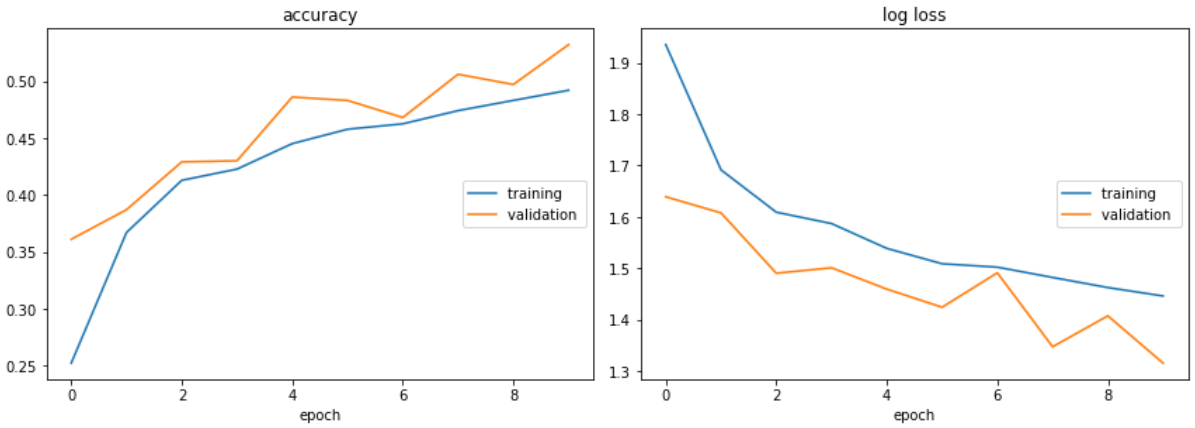
In [334]:
```
#Load AlexNet
model_ft = alexnet()

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft.load_state_dict(torch.load('models/Alex_sgd/model_11_epoch.pt'
))
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: 21.763010025024414.5.
Test Loss: 2.8026 Acc: 0.4463

Test complete in 0m 9s
```

Alexnet-Relu - optimizer:Adam

In [304]:
```
#Load AlexNet
model_ft = alexnet()

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 [305]:
```
train_model("Alex_adam",model_ft, dataloaders, dataset_sizes, criterion,
optimizer_ft, num_epochs=10)
```




```
accuracy
        training                (min:    0.252, max:    0.492, cur:
0.492)
        validation              (min:    0.361, max:    0.532, cur:
0.532)
log loss
        training                (min:    1.446, max:    1.935, cur:
1.446)
        validation              (min:    1.316, max:    1.639, cur:
1.316)
Train Loss: 1.4463 Acc: 0.4919
Val Loss: 1.3156 Acc: 0.5320

Training complete in 24m 48s
Best Validation Accuracy: 0.532, Epoch: 10
```

In [306]:
```
#Load AlexNet
model_ft = alexnet()

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft.load_state_dict(torch.load('models/Alex_adam/model_10_epoch.pt'
))
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
)
```

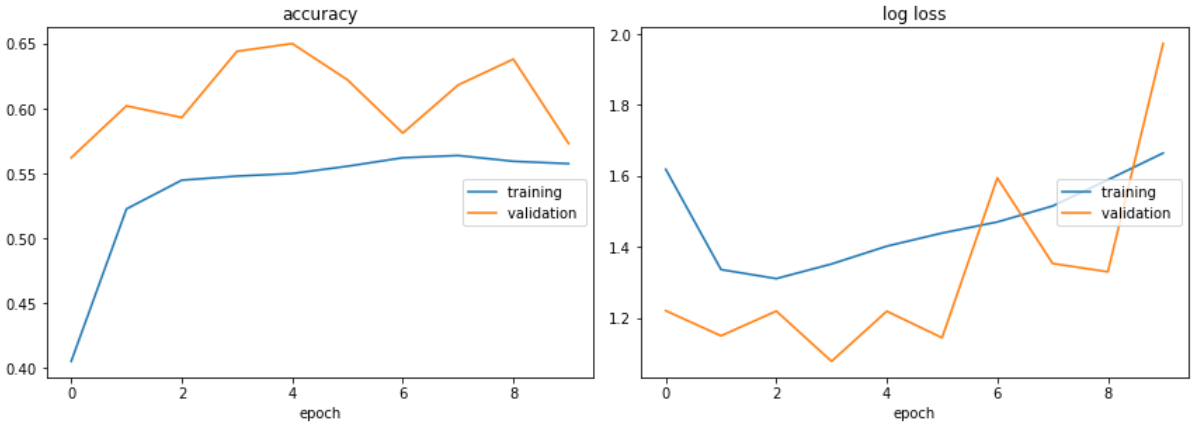
```
Iteration: 2500/2500, Loss: 15.240313529968262.
Test Loss: 2.4420 Acc: 0.2305

Test complete in 0m 10s
```

Alexnet- Tanh

In [288]:
```
class AlexNet(nn.Module):

    def __init__(self, num_classes=10):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=2, padding=1),
            nn.Tanh(),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(64, 192, kernel_size=3, padding=1),
            nn.Tanh(),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.Tanh(),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.Tanh(),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.Tanh(),
            nn.MaxPool2d(kernel_size=2),
        )
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 2 * 2, 4096),
            nn.Tanh(),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.Tanh(),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), 256 * 2 * 2)
        x = self.classifier(x)
        return x
```

In [273]:
```
#Load AlexNet
model_ft = alexnet()

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)
```

In [274]:
```
train_model("Alex_tanh",model_ft, dataloaders, dataset_sizes, criterion,
optimizer_ft, num_epochs=10)
```




```
accuracy
        training                 (min:    0.405, max:    0.564, cur:
0.557)
        validation               (min:    0.562, max:    0.650, cur:
0.573)
log loss
        training                 (min:    1.310, max:    1.664, cur:
1.664)
        validation               (min:    1.077, max:    1.973, cur:
1.973)
Train Loss: 1.6640 Acc: 0.5574
Val Loss: 1.9726 Acc: 0.5730

Training complete in 15m 49s
Best Validation Accuracy: 0.65, Epoch: 5
```

In [289]:
```
#Load AlexNet
model_ft = alexnet()

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft.load_state_dict(torch.load('models/Alex_tanh/model_5_epoch.pt'
))
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
)
```

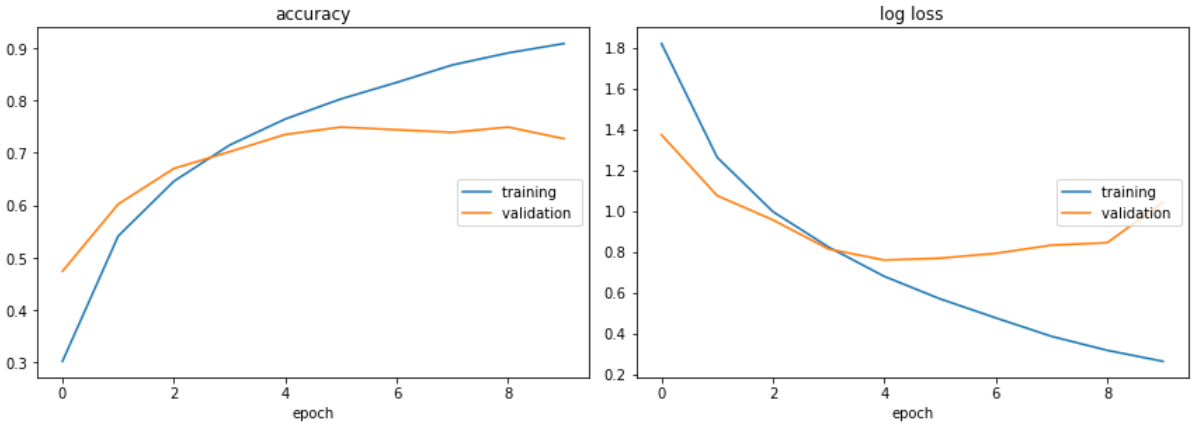
```
Iteration: 2500/2500, Loss: 11.6156644821167...
Test Loss: 2.2321 Acc: 0.3937

Test complete in 0m 10s
```

Alexnet- LeakvRelu

In [280]:
```
class AlexNet(nn.Module):

    def __init__(self, num_classes=10):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=2, padding=1),
            nn.LeakyReLU(inplace = True),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(64, 192, kernel_size=3, padding=1),
            nn.LeakyReLU(inplace = True),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.LeakyReLU(inplace = True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.LeakyReLU(inplace = True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.LeakyReLU(inplace = True),
            nn.MaxPool2d(kernel_size=2),
        )
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 2 * 2, 4096),
            nn.LeakyReLU(inplace = True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.LeakyReLU(inplace = True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), 256 * 2 * 2)
        x = self.classifier(x)
        return x
```

In [281]:
```
#Load AlexNet
model_ft = alexnet()

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)
```

In [282]:
```
train_model("Alex_leaky",model_ft, dataloaders, dataset_sizes, criterion
, optimizer_ft, num_epochs=10)
```




```
accuracy
        training                (min:    0.302, max:    0.908, cur:
0.908)
        validation              (min:    0.474, max:    0.749, cur:
0.727)
log loss
        training                (min:    0.262, max:    1.821, cur:
0.262)
        validation              (min:    0.759, max:    1.373, cur:
1.041)
Train Loss: 0.2623 Acc: 0.9085
Val Loss: 1.0414 Acc: 0.7270

Training complete in 15m 52s
Best Validation Accuracy: 0.749, Epoch: 6
```

In [286]:
```
#Load AlexNet
model_ft = alexnet()

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft.load_state_dict(torch.load('models/Alex_leaky/model_6_epoch.pt'
))
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
)
```

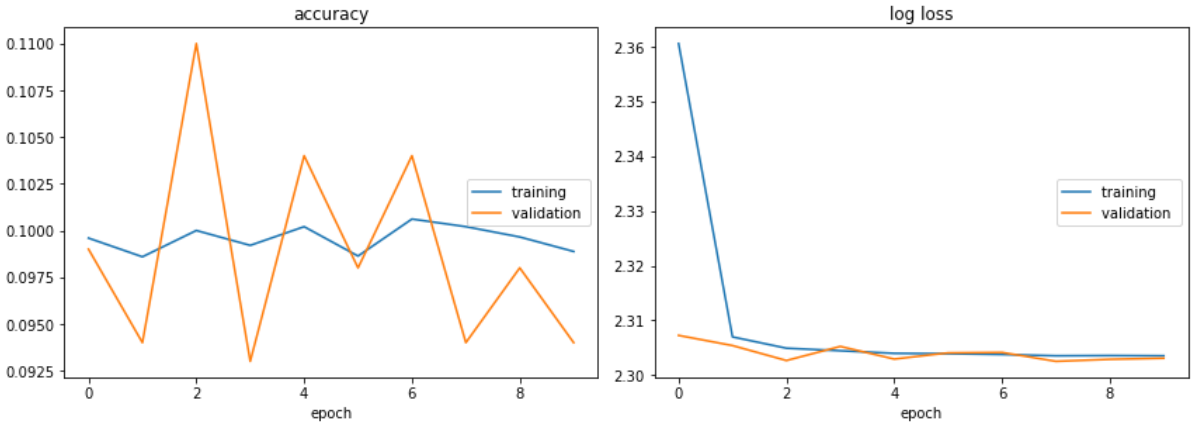
```
Iteration: 2500/2500, Loss: 12.675323486328125..
Test Loss: 1.7513 Acc: 0.4848

Test complete in 0m 9s
```

Alexnet- Sigmoid

In [291]:
```
class AlexNet(nn.Module):

    def __init__(self, num_classes=10):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=2, padding=1),
            nn.Sigmoid(),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(64, 192, kernel_size=3, padding=1),
            nn.Sigmoid(),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.Sigmoid(),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.Sigmoid(),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.Sigmoid(),
            nn.MaxPool2d(kernel_size=2),
        )
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 2 * 2, 4096),
            nn.Sigmoid(),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.Sigmoid(),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), 256 * 2 * 2)
        x = self.classifier(x)
        return x
```

In [292]:
```
#Load AlexNet
model_ft = alexnet()

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)
```

In [293]:
```
train_model("Alex_sigmoid",model_ft, dataloaders, dataset_sizes, criterion, optimizer_ft, num_epochs=20)
```




```
accuracy
        training                 (min:    0.099, max:    0.101, cur:
0.099)
        validation               (min:    0.093, max:    0.110, cur:
0.094)
log loss
        training                 (min:    2.304, max:    2.361, cur:
2.304)
        validation               (min:    2.303, max:    2.307, cur:
2.303)
Train Loss: 2.3036 Acc: 0.0989
Val Loss: 2.3031 Acc: 0.0940

Training complete in 16m 8s
Best Validation Accuracy: 0.11, Epoch: 3
```

In [294]:
```
#Load AlexNet
model_ft = alexnet()

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft.load_state_dict(torch.load('models/Alex_sigmoid/model_3_epoch.pt'))
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)
```

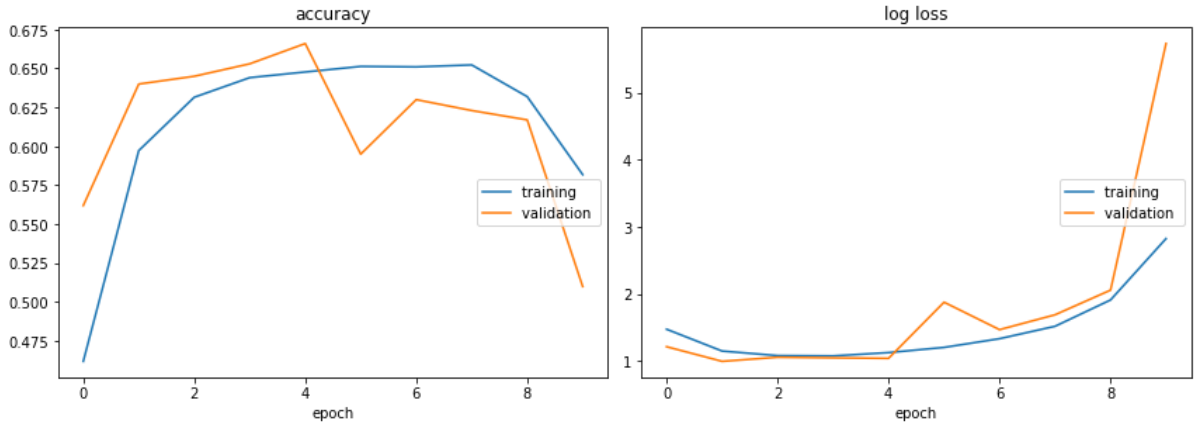
```
Iteration: 2500/2500, Loss: 9.132913589477539.
Test Loss: 2.3040 Acc: 0.1000

Test complete in 0m 10s
```

Alexnet- Elu

In [308]:
```
class AlexNet(nn.Module):

    def __init__(self, num_classes=10):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=2, padding=1),
            nn.ELU(),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(64, 192, kernel_size=3, padding=1),
            nn.ELU(),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ELU(),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ELU(),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ELU(),
            nn.MaxPool2d(kernel_size=2),
        )
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 2 * 2, 4096),
            nn.ELU(),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ELU(),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), 256 * 2 * 2)
        x = self.classifier(x)
        return x
```

In [309]:
```
#Load AlexNet
model_ft = alexnet()

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)
```

In [310]:
```
train_model("Alex_elu",model_ft, dataloaders, dataset_sizes, criterion,
optimizer_ft, num_epochs=10)
```




```
accuracy
        training                 (min:    0.462, max:    0.652, cur:
0.582)
        validation               (min:    0.510, max:    0.666, cur:
0.510)
log loss
        training                 (min:    1.075, max:    2.823, cur:
2.823)
        validation               (min:    0.994, max:    5.730, cur:
5.730)
Train Loss: 2.8231 Acc: 0.5818
Val Loss: 5.7301 Acc: 0.5100

Training complete in 15m 59s
Best Validation Accuracy: 0.666, Epoch: 5
```

In [311]:
```
#Load AlexNet
model_ft = alexnet()

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft.load_state_dict(torch.load('models/Alex_elu/model_5_epoch.pt'))
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
)
```

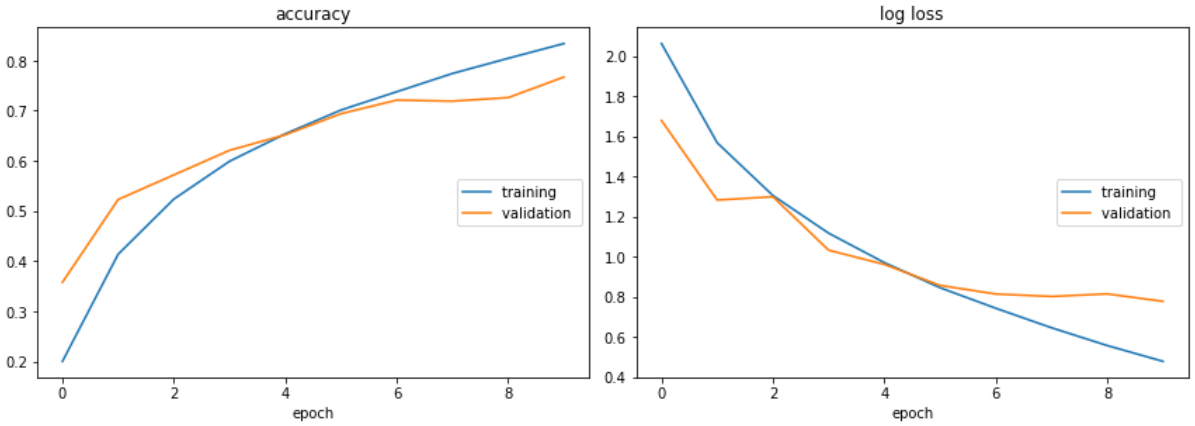
```
Iteration: 2500/2500, Loss: 9.334567070007324..
Test Loss: 2.0060 Acc: 0.3979

Test complete in 0m 10s
```

Alexnet- average pooling

In [295]:
```
class AlexNet(nn.Module):

    def __init__(self, num_classes=10):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=2, padding=1),
            nn.ReLU(inplace=True),
            nn.AvgPool2d(kernel_size=2),
            nn.Conv2d(64, 192, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.AvgPool2d(kernel_size=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.AvgPool2d(kernel_size=2),
        )
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 2 * 2, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), 256 * 2 * 2)
        x = self.classifier(x)
        return x
```

In [296]:
```
#Load AlexNet
model_ft = alexnet()

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)
```

In [297]:
```
train_model("Alexnet_avgpool",model_ft, dataloaders, dataset_sizes, crit
erion, optimizer_ft, num_epochs=10)
```




```
accuracy
        training                (min:    0.200, max:    0.834, cur:
0.834)
        validation              (min:    0.358, max:    0.767, cur:
0.767)
log loss
        training                (min:    0.478, max:    2.061, cur:
0.478)
        validation              (min:    0.777, max:    1.678, cur:
0.777)
Train Loss: 0.4777 Acc: 0.8338
Val Loss: 0.7768 Acc: 0.7670

Training complete in 15m 55s
Best Validation Accuracy: 0.767, Epoch: 10
```

In [298]:
```
#Load AlexNet
model_ft = alexnet()

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft.load_state_dict(torch.load('models/Alexnet_avgpool/model_10_epo
ch.pt'))
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: 15.447891235351562..
Test Loss: 1.9702 Acc: 0.4470

Test complete in 0m 10s
```

Fine tune- RandomHorizontalFlip

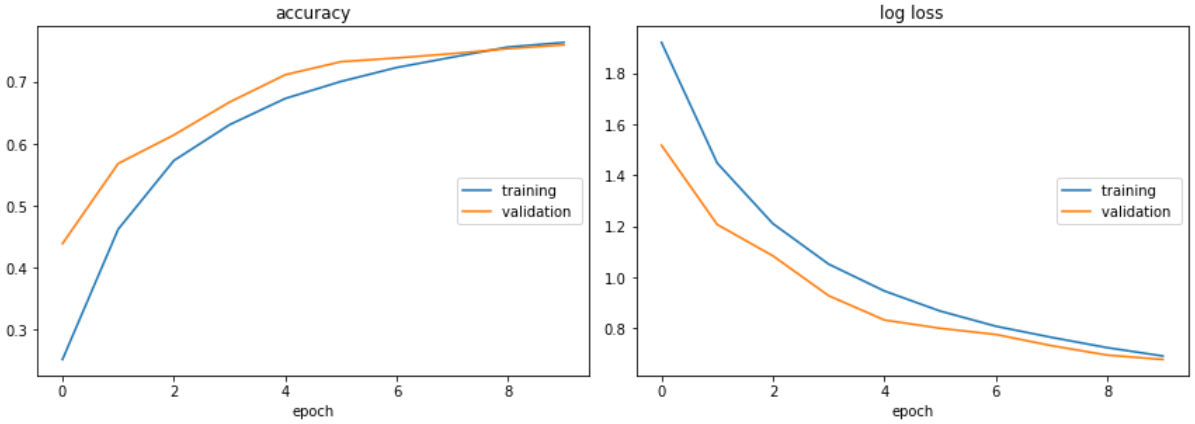
In [221]:
```
class AlexNet(nn.Module):

    def __init__(self, num_classes=10):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, stride=2, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(64, 192, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2),
        )
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 2 * 2, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), 256 * 2 * 2)
        x = self.classifier(x)
        return x
```

In [261]:
```
trainloader, valloader = get_train_valid_loader('./data', batch_size=4,
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}
```

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

In [223]:
```
#Load AlexNet
model_ft = alexnet()

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)
```

In [224]:
```
#Train
train_model("aug1",model_ft, dataloaders, dataset_sizes, criterion, opti
mizer_ft, num_epochs=10)
```




```
accuracy
        training                (min:    0.252, max:    0.763, cur:
0.763)
        validation              (min:    0.439, max:    0.759, cur:
0.759)
log loss
        training                (min:    0.690, max:    1.920, cur:
0.690)
        validation              (min:    0.677, max:    1.517, cur:
0.677)
Train Loss: 0.6902 Acc: 0.7630
Val Loss: 0.6765 Acc: 0.7590

Training complete in 15m 44s
Best Validation Accuracy: 0.759, Epoch: 10
```

In [262]:
```
#Load AlexNet
model_ft = alexnet()

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft.load_state_dict(torch.load('models/aug1/model_10_epoch.pt'))
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: 13.029035568237305..
Test Loss: 1.9625 Acc: 0.4534

Test complete in 0m 9s
```

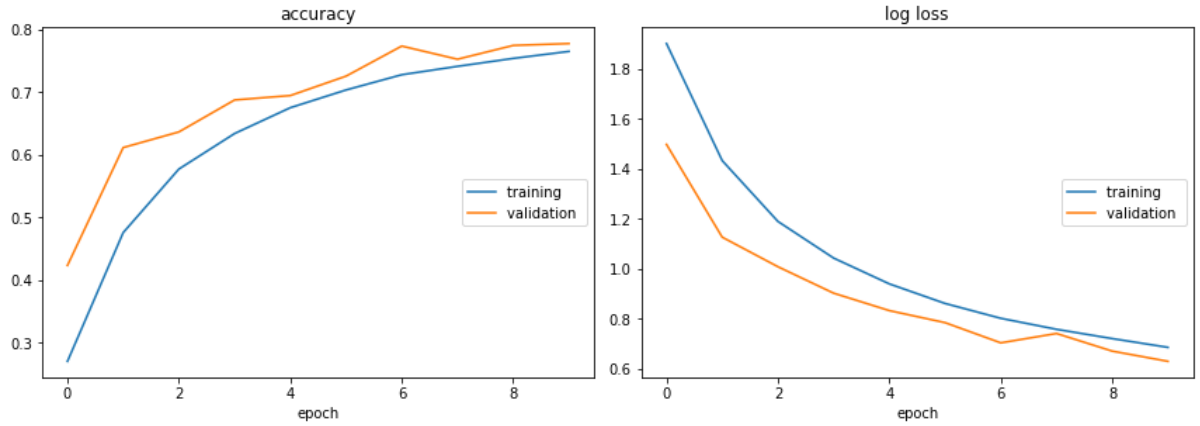
Finetune- RandomVerticalFlip

In [263]:
```
trainloader, valloader = get_train_valid_loader('./data', batch_size=4,
augment=2, random_seed=34567,
                                                valid_size=0.02, shuffle
=True, num_workers=2, pin_memory=True)

dataloaders = {'train': trainloader, 'val': valloader, 'test': testloade
r}
```

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

In [226]:
```
#Load AlexNet
model_ft = alexnet()

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)
```

In [227]:
```
#Train
train_model("aug2",model_ft, dataloaders, dataset_sizes, criterion, opti
mizer_ft, num_epochs=10)
```




```
accuracy
        training                (min:    0.270, max:    0.764, cur:
0.764)
        validation              (min:    0.423, max:    0.777, cur:
0.777)
log loss
        training                (min:    0.685, max:    1.902, cur:
0.685)
        validation              (min:    0.629, max:    1.498, cur:
0.629)
Train Loss: 0.6850 Acc: 0.7644
Val Loss: 0.6290 Acc: 0.7770

Training complete in 15m 49s
Best Validation Accuracy: 0.777, Epoch: 10
```

In [264]:
```
#Load AlexNet
model_ft = alexnet()

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft.load_state_dict(torch.load('models/aug2/model_10_epoch.pt'))
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: 14.188934326171875.
Test Loss: 1.8152 Acc: 0.4939

Test complete in 0m 9s
```

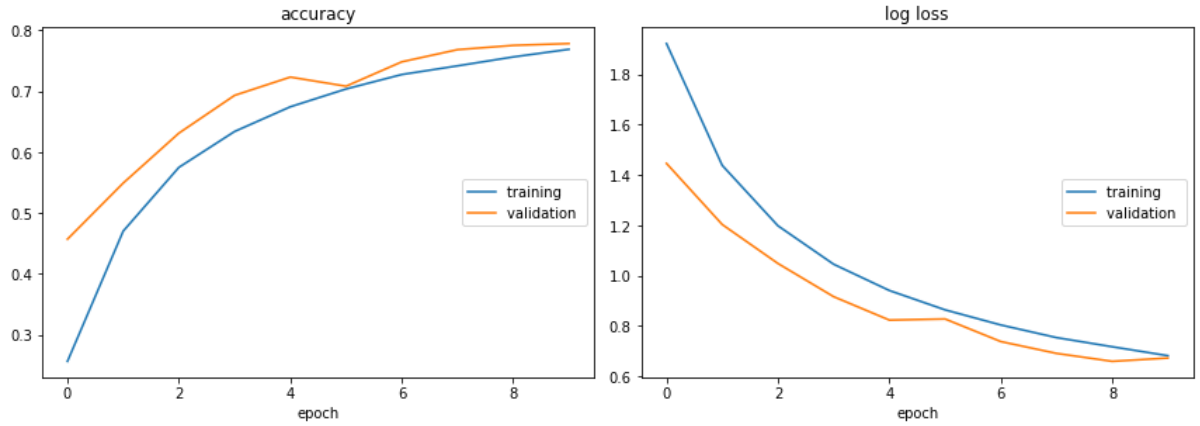
Fine tune- RandomRotation

In [265]:
```
trainloader, valloader = get_train_valid_loader('./data', batch_size=4,
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}
```

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

In [229]:
```
#Load AlexNet
model_ft = alexnet()

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)
```

In [230]:
```
#Train
train_model("aug3",model_ft, dataloaders, dataset_sizes, criterion, opti
mizer_ft, num_epochs=10)
```




```
accuracy
        training                (min:    0.257, max:    0.769, cur:
0.769)
        validation              (min:    0.457, max:    0.778, cur:
0.778)
log loss
        training                (min:    0.681, max:    1.922, cur:
0.681)
        validation              (min:    0.659, max:    1.446, cur:
0.672)
Train Loss: 0.6812 Acc: 0.7686
Val Loss: 0.6720 Acc: 0.7780

Training complete in 15m 58s
Best Validation Accuracy: 0.778, Epoch: 10
```

In [266]:
```
#Load AlexNet
model_ft = alexnet()

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft.load_state_dict(torch.load('models/aug3/model_10_epoch.pt'))
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: 13.035179138183594..
Test Loss: 2.1031 Acc: 0.4322

Test complete in 0m 9s
```

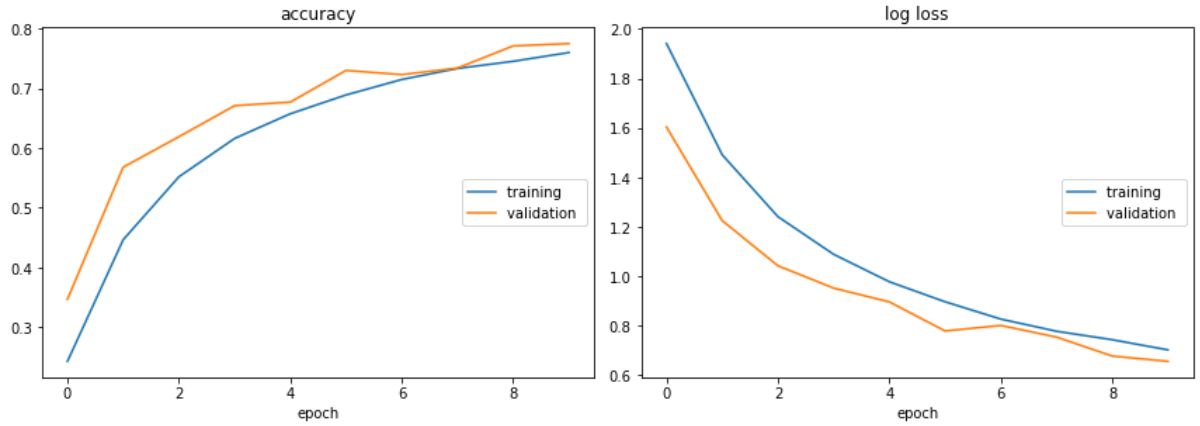
Fine tune - Combination

In [267]:
```
trainloader, valloader = get_train_valid_loader('./data', batch_size=4,
augment=4, random_seed=34567,
                                                valid_size=0.02, shuffle
=True, num_workers=2, pin_memory=True)

dataloaders = {'train': trainloader, 'val': valloader, 'test': testloade
r}
```

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

In [232]:
```
#Load AlexNet
model_ft = alexnet()

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)
```

In [233]:
```
#Train
train_model("aug4",model_ft, dataloaders, dataset_sizes, criterion, opti
mizer_ft, num_epochs=10)
```




```
accuracy
        training                (min:    0.243, max:    0.760, cur:
0.760)
        validation              (min:    0.347, max:    0.775, cur:
0.775)
log loss
        training                (min:    0.703, max:    1.941, cur:
0.703)
        validation              (min:    0.656, max:    1.604, cur:
0.656)
Train Loss: 0.7026 Acc: 0.7599
Val Loss: 0.6564 Acc: 0.7750

Training complete in 16m 2s
Best Validation Accuracy: 0.775, Epoch: 10
```

In [268]:
```
#Load AlexNet
model_ft = alexnet()

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft.load_state_dict(torch.load('models/aug4/model_10_epoch.pt'))
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: 15.442499160766602..
Test Loss: 1.6911 Acc: 0.4895

Test complete in 0m 9s
```