PART WISE

```
from google.colab import files
files.upload()  # Select your kaggle.json file
```

```
<IPython.core.display.HTML object>

Saving kaggle.json to kaggle.json

{'kaggle.json':
b'{"username":"subhankardash1603","key":"d5841ce4d3236107f9c550b31ce62
f59"}'}
```

```
!pip install -q kaggle
```

```
import os
os.makedirs('/root/.kaggle', exist_ok=True)
!mv kaggle.json /root/.kaggle/
!chmod 600 /root/.kaggle/kaggle.json
```

```
!kaggle datasets download -d alessiocorrado99/animals10
```

```
Dataset URL:
https://www.kaggle.com/datasets/alessiocorrado99/animals10
License(s): GPL-2.0
Downloading animals10.zip to /content
 99% 580M/586M [00:01<00:00, 302MB/s]
100% 586M/586M [00:01<00:00, 363MB/s]
```

```
import zipfile
```

```
with zipfile.ZipFile("animals10.zip", 'r') as zip_ref:
    zip_ref.extractall("animals10")
```

```
import os
os.listdir("animals10/raw-img")
```

```
['scoiattolo',
 'gatto',
 'mucca',
 'cavallo',
 'pecora',
 'cane',
 'farfalla',
 'gallina',
 'elefante',
 'ragno']
```

```
# Check GPU access
import tensorflow as tf
```

```
device_name = tf.test.gpu_device_name()
if not device_name:
    raise SystemError('GPU device not found. Go to Runtime > Change runtime type > GPU.')
print(f'□ GPU available at: {device_name}')
```

```
□ GPU available at: /device:GPU:0
```

Import Libraries and Set Dataset Path

```
import os
import time
import zipfile
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import numpy as np
from torch.utils.data import DataLoader
from sklearn.metrics import accuracy_score

# Set data directories
data_dir = "animals10/raw-img"
```

Prepare Transforms, Dataset Splits, and Data Loaders

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

dataset = datasets.ImageFolder(data_dir, transform=transform)
train_size = int(0.8 * len(dataset))
val_size = len(dataset) - train_size
train_dataset, val_dataset = torch.utils.data.random_split(dataset, [train_size, val_size])

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)

class_names = dataset.classes
num_classes = len(class_names)
```

Define ZFNet, VGG16, and GoogLeNet Architectures

```
class ZFNet(nn.Module):
    def __init__(self, num_classes):
        super(ZFNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 96, kernel_size=7, stride=2, padding=1),  #
Output: (96, 109, 109)
            nn.ReLU(), nn.MaxPool2d(3, 2),                          #
Output: (96, 54, 54)
            nn.Conv2d(96, 256, kernel_size=5, padding=2),          #
Output: (256, 54, 54)
            nn.ReLU(), nn.MaxPool2d(3, 2),                          #
Output: (256, 26, 26)
            nn.Conv2d(256, 384, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(384, 384, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(), nn.MaxPool2d(3, 2)                           #
Output: depends on input size
        )

        # Dynamically determine the size after feature extraction
        with torch.no_grad():
            dummy_input = torch.zeros(1, 3, 224, 224)
            dummy_output = self.features(dummy_input)
            self.flattened_size = dummy_output.view(1, -1).shape[1]

        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(self.flattened_size, 4096),
            nn.ReLU(),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(),
            nn.Linear(4096, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        return self.classifier(x)


def build_vgg16(num_classes):
    model = models.vgg16(weights=models.VGG16_Weights.DEFAULT)
    for param in model.features.parameters():
        param.requires_grad = False
    model.classifier[6] = nn.Linear(4096, num_classes)
    return model.to(device)
```

```
def build_googlenet(num_classes):
    model = models.googlenet(weights=models.GoogLeNet_Weights.DEFAULT)
    for param in model.parameters():
        param.requires_grad = False
    model.fc = nn.Linear(model.fc.in_features, num_classes)
    return model.to(device)
```

Define Model Training Function (train_model)

```
def train_model(model, model_name, train_loader, val_loader,
epochs=3):
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    train_acc, val_acc, train_loss, val_loss = [], [], [], []

    since = time.time()

    for epoch in range(epochs):
        model.train()
        running_loss, running_corrects = 0.0, 0

        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            _, preds = torch.max(outputs, 1)
            loss.backward()
            optimizer.step()

            running_loss += loss.item() * inputs.size(0)
            running_corrects += torch.sum(preds == labels.data)

        epoch_loss = running_loss / len(train_loader.dataset)
        epoch_acc = running_corrects.double() /
len(train_loader.dataset)
        train_loss.append(epoch_loss)
        train_acc.append(epoch_acc.item())

        # Validation
        model.eval()
        val_running_loss, val_running_corrects = 0.0, 0
        with torch.no_grad():
            for inputs, labels in val_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                _, preds = torch.max(outputs, 1)
                val_running_loss += loss.item() * inputs.size(0)
```

```
                val_running_corrects += torch.sum(preds ==
labels.data)

        val_epoch_loss = val_running_loss / len(val_loader.dataset)
        val_epoch_acc = val_running_corrects.double() /
len(val_loader.dataset)
        val_loss.append(val_epoch_loss)
        val_acc.append(val_epoch_acc.item())

        print(f"Epoch {epoch+1}/{epochs} => "
              f"Train Acc: {epoch_acc:.4f}, Val Acc:
{val_epoch_acc:.4f}")

    time_elapsed = time.time() - since
    print(f"Training complete in {time_elapsed // 60:.0f}m
{time_elapsed % 60:.0f}s")

    return model, {'train_loss': train_loss, 'val_loss': val_loss,
                   'train_acc': train_acc, 'val_acc': val_acc},
val_acc[-1], time_elapsed
```

Train ZFNet, VGG16, and GoogLeNet Models

```
models_to_train = {
    'ZFNet': ZFNet(num_classes).to(device),
    'VGG16': build_vgg16(num_classes),
    'GoogLeNet': build_googlenet(num_classes)
}


results = {}

for name, model in models_to_train.items():
    print(f"\nTraining {name}")
    trained_model, history, val_acc, training_time =
train_model(model, name, train_loader, val_loader, epochs=10)
    results[name] = {
        'model': trained_model,
        'history': history,
        'val_accuracy': val_acc,
        'training_time': training_time
    }
```

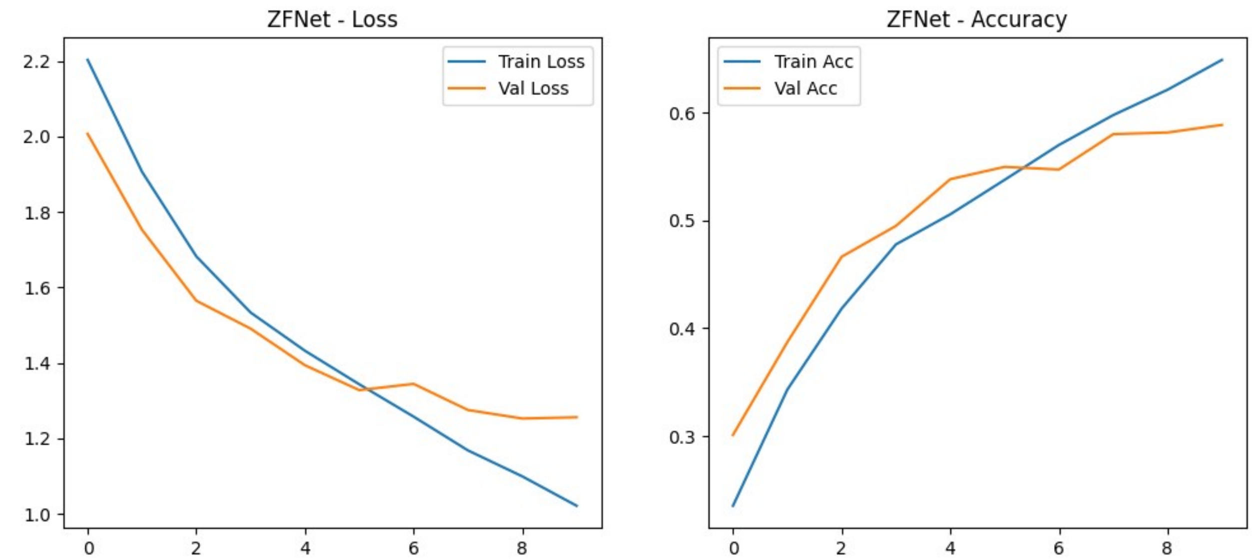
```
Training ZFNet
Epoch 1/10 => Train Acc: 0.2352, Val Acc: 0.3010
Epoch 2/10 => Train Acc: 0.3430, Val Acc: 0.3871
Epoch 3/10 => Train Acc: 0.4183, Val Acc: 0.4664
Epoch 4/10 => Train Acc: 0.4779, Val Acc: 0.4950
```

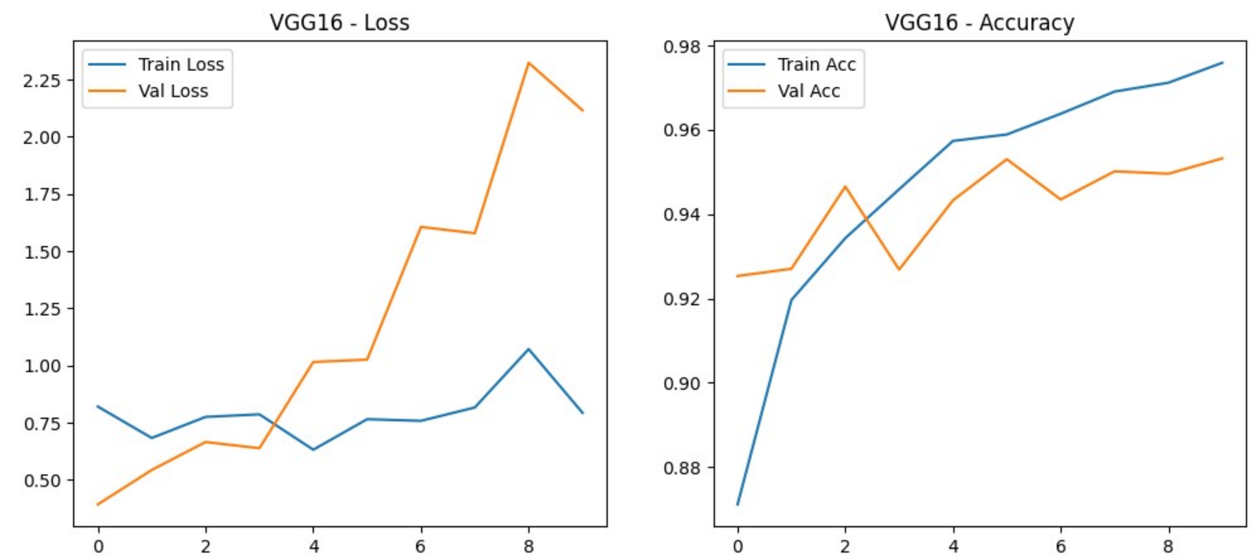
```
Epoch 5/10 => Train Acc: 0.5058, Val Acc: 0.5384
Epoch 6/10 => Train Acc: 0.5378, Val Acc: 0.5498
Epoch 7/10 => Train Acc: 0.5702, Val Acc: 0.5474
Epoch 8/10 => Train Acc: 0.5979, Val Acc: 0.5802
Epoch 9/10 => Train Acc: 0.6214, Val Acc: 0.5817
Epoch 10/10 => Train Acc: 0.6491, Val Acc: 0.5888
Training complete in 29m 47s

Training VGG16
Epoch 1/10 => Train Acc: 0.8712, Val Acc: 0.9253
Epoch 2/10 => Train Acc: 0.9196, Val Acc: 0.9270
Epoch 3/10 => Train Acc: 0.9343, Val Acc: 0.9465
Epoch 4/10 => Train Acc: 0.9459, Val Acc: 0.9269
Epoch 5/10 => Train Acc: 0.9574, Val Acc: 0.9433
Epoch 6/10 => Train Acc: 0.9589, Val Acc: 0.9530
Epoch 7/10 => Train Acc: 0.9638, Val Acc: 0.9435
Epoch 8/10 => Train Acc: 0.9691, Val Acc: 0.9502
Epoch 9/10 => Train Acc: 0.9712, Val Acc: 0.9496
Epoch 10/10 => Train Acc: 0.9759, Val Acc: 0.9532
Training complete in 39m 25s

Training GoogLeNet
Epoch 1/10 => Train Acc: 0.9008, Val Acc: 0.9473
Epoch 2/10 => Train Acc: 0.9359, Val Acc: 0.9515
Epoch 3/10 => Train Acc: 0.9377, Val Acc: 0.9572
Epoch 4/10 => Train Acc: 0.9422, Val Acc: 0.9576
Epoch 5/10 => Train Acc: 0.9443, Val Acc: 0.9561
Epoch 6/10 => Train Acc: 0.9466, Val Acc: 0.9557
Epoch 7/10 => Train Acc: 0.9454, Val Acc: 0.9589
Epoch 8/10 => Train Acc: 0.9455, Val Acc: 0.9578
Epoch 9/10 => Train Acc: 0.9487, Val Acc: 0.9513
Epoch 10/10 => Train Acc: 0.9475, Val Acc: 0.9608
Training complete in 18m 56s
```

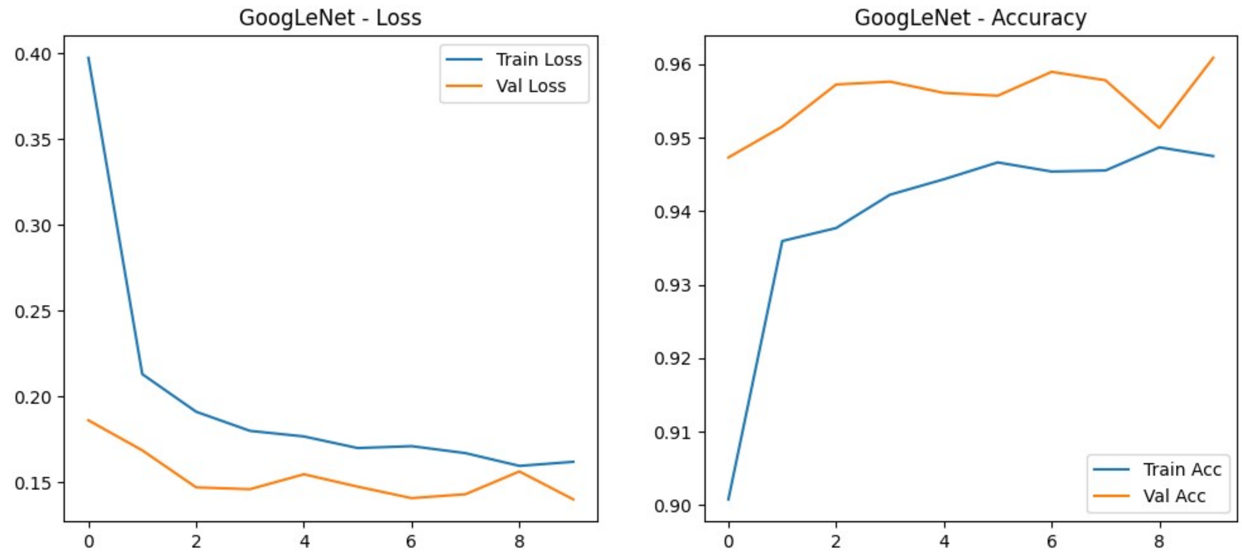
Plot Training and Validation Accuracy/Loss for Each Model

```python
def plot_history(history, model_name):
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(history['train_loss'], label='Train Loss')
    plt.plot(history['val_loss'], label='Val Loss')
    plt.title(f'{model_name} - Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history['train_acc'], label='Train Acc')
    plt.plot(history['val_acc'], label='Val Acc')
    plt.title(f'{model_name} - Accuracy')
    plt.legend()
```

```
    plt.show()

for name in results:
    plot_history(results[name]['history'], name)
```

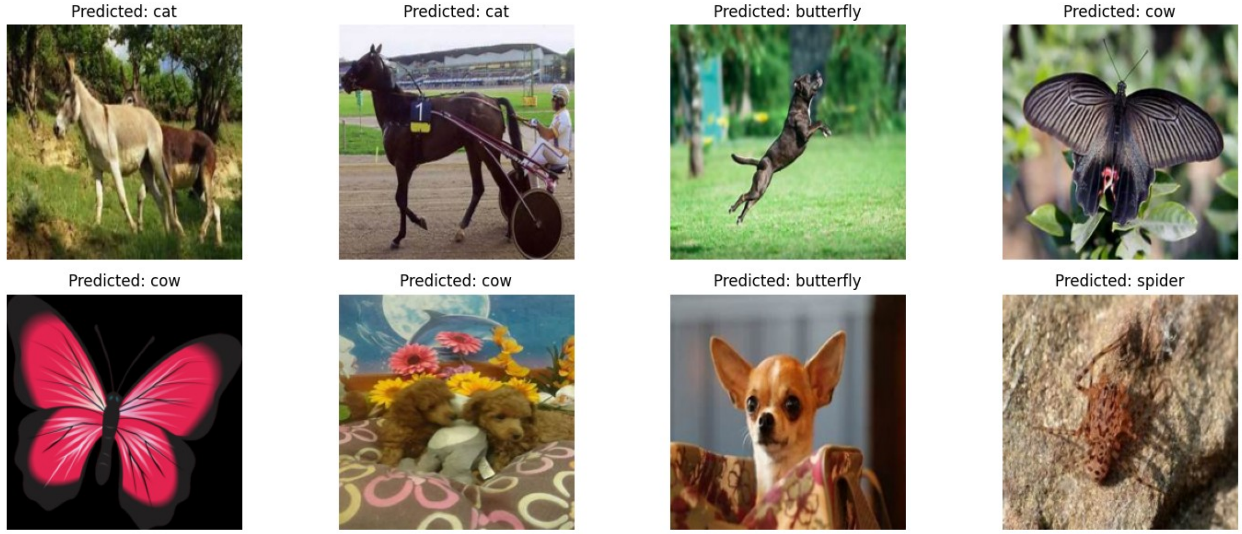






Visualize Sample Predictions from GoogLeNet

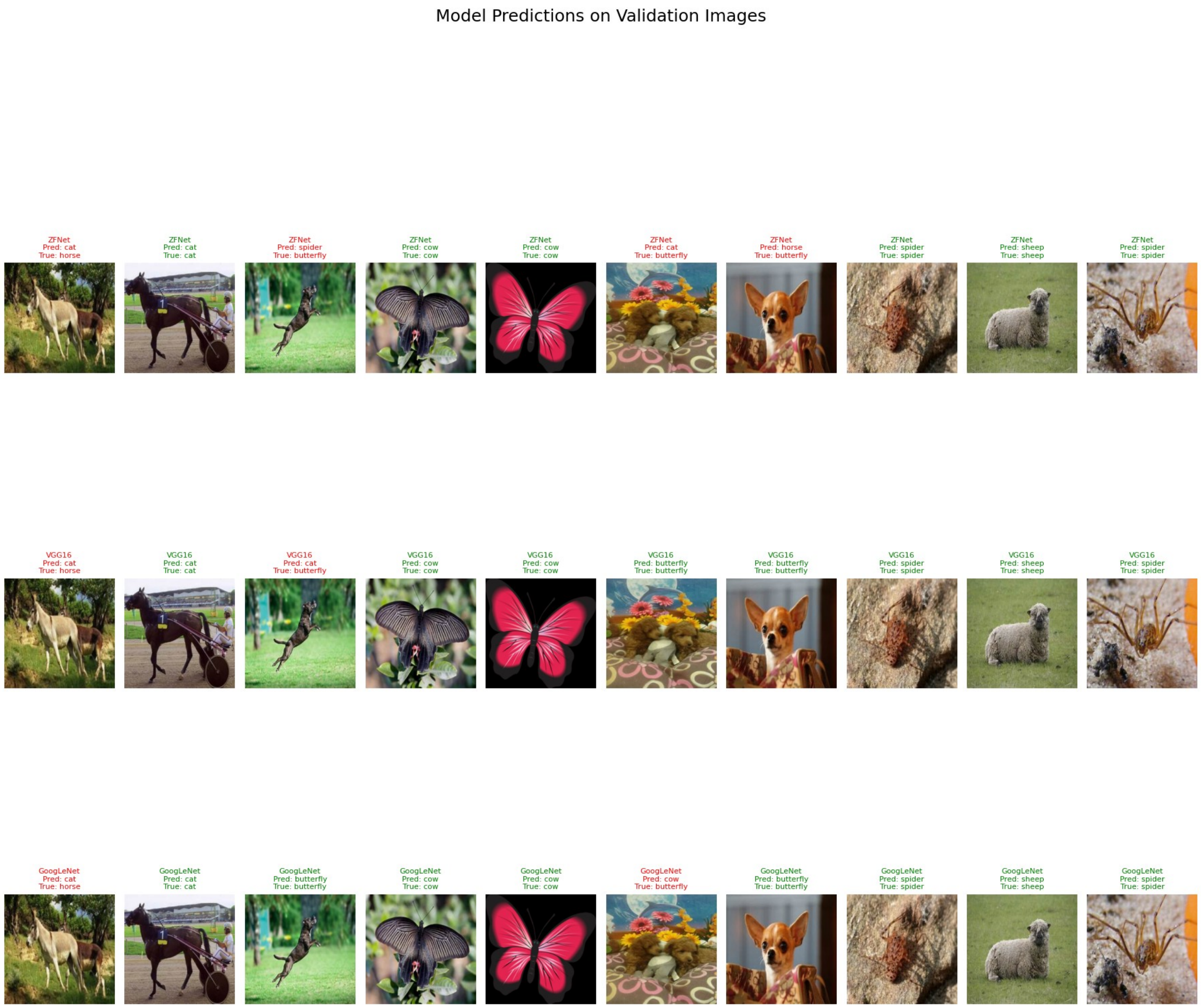
```
def imshow(inp, title=None):
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = np.clip((inp * std + mean), 0, 1)
    plt.imshow(inp)
    if title:
        plt.title(title)
    plt.axis('off')

# Show predictions from GoogLeNet
model = results['GoogLeNet']['model']
model.eval()
inputs, classes = next(iter(val_loader))
inputs = inputs.to(device)
outputs = model(inputs)
_, preds = torch.max(outputs, 1)

plt.figure(figsize=(15, 6))
for idx in range(8):
    ax = plt.subplot(2, 4, idx+1)
    imshow(inputs.cpu().data[idx])
    ax.set_title(f"Predicted: {class_names[preds[idx]]}")
plt.tight_layout()
plt.show()
```



Visual Comparison of Predictions from ZFNet, VGG16, and GoogLeNet

```
def imshow(inp, title=None):
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = np.clip((inp * std + mean), 0, 1)
    plt.imshow(inp)
    if title:
        plt.title(title)
    plt.axis('off')

# Get a fixed batch of validation images
inputs, true_classes = next(iter(val_loader))
inputs = inputs.to(device)
true_classes = true_classes.to(device)

# Display predictions from all 3 models
plt.figure(figsize=(18, 18))
num_models = len(results)
num_samples = 10

accuracies = {}

for i, (name, result) in enumerate(results.items()):
    model = result['model']
    model.eval()
    with torch.no_grad():
        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)

    correct_preds = (preds[:num_samples] ==
true_classes[:num_samples]).sum().item()
```

```python
    accuracy_percent = correct_preds / num_samples * 100
    accuracies[name] = accuracy_percent

    for j in range(num_samples):
        idx = i * num_samples + j + 1
        ax = plt.subplot(num_models, num_samples, idx)
        imshow(inputs.cpu().data[j])
        pred_label = class_names[preds[j]]
        true_label = class_names[true_classes[j]]
        title_color = 'green' if pred_label == true_label else 'red'
        ax.set_title(f"{name}\nPred: {pred_label}\nTrue:
{true_label}", color=title_color, fontsize=8)
        ax.axis('off')

plt.suptitle("Model Predictions on Validation Images", fontsize=18)
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()

# Print accuracy summary
print("Prediction Accuracy on 10 Samples:\n")
for model_name, acc in accuracies.items():
    print(f"{model_name}: {acc:.2f}%")
```

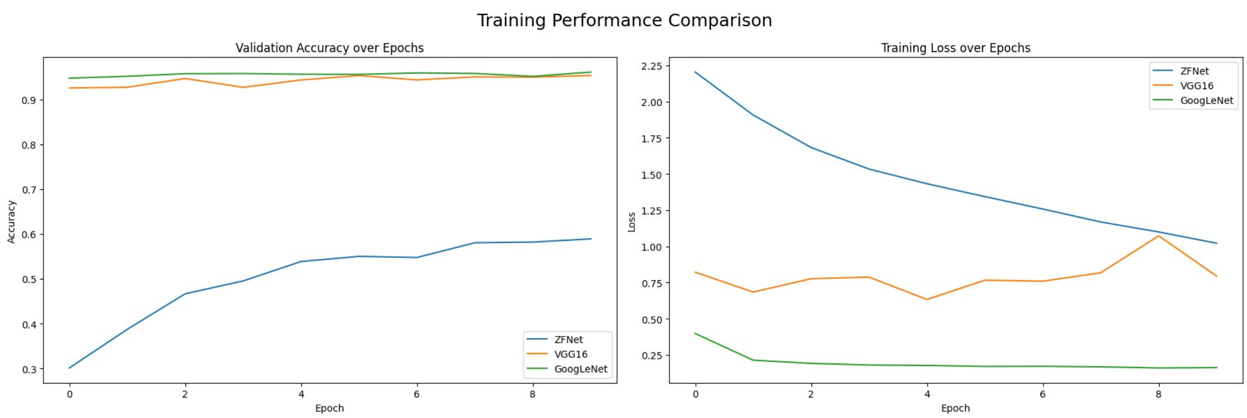



```
Prediction Accuracy on 10 Samples:

ZFNet: 60.00%
VGG16: 80.00%
GoogLeNet: 80.00%
```

Plot Training Loss and Validation Accuracy Comparison for All Models

```python
def plot_training_history(results):
    plt.figure(figsize=(18, 6))

    # Accuracy
    plt.subplot(1, 2, 1)
    for name, result in results.items():
        acc = result['history']['val_acc']
        plt.plot(acc, label=f'{name}')
    plt.title('Validation Accuracy over Epochs')
    plt.xlabel('Epoch')
```

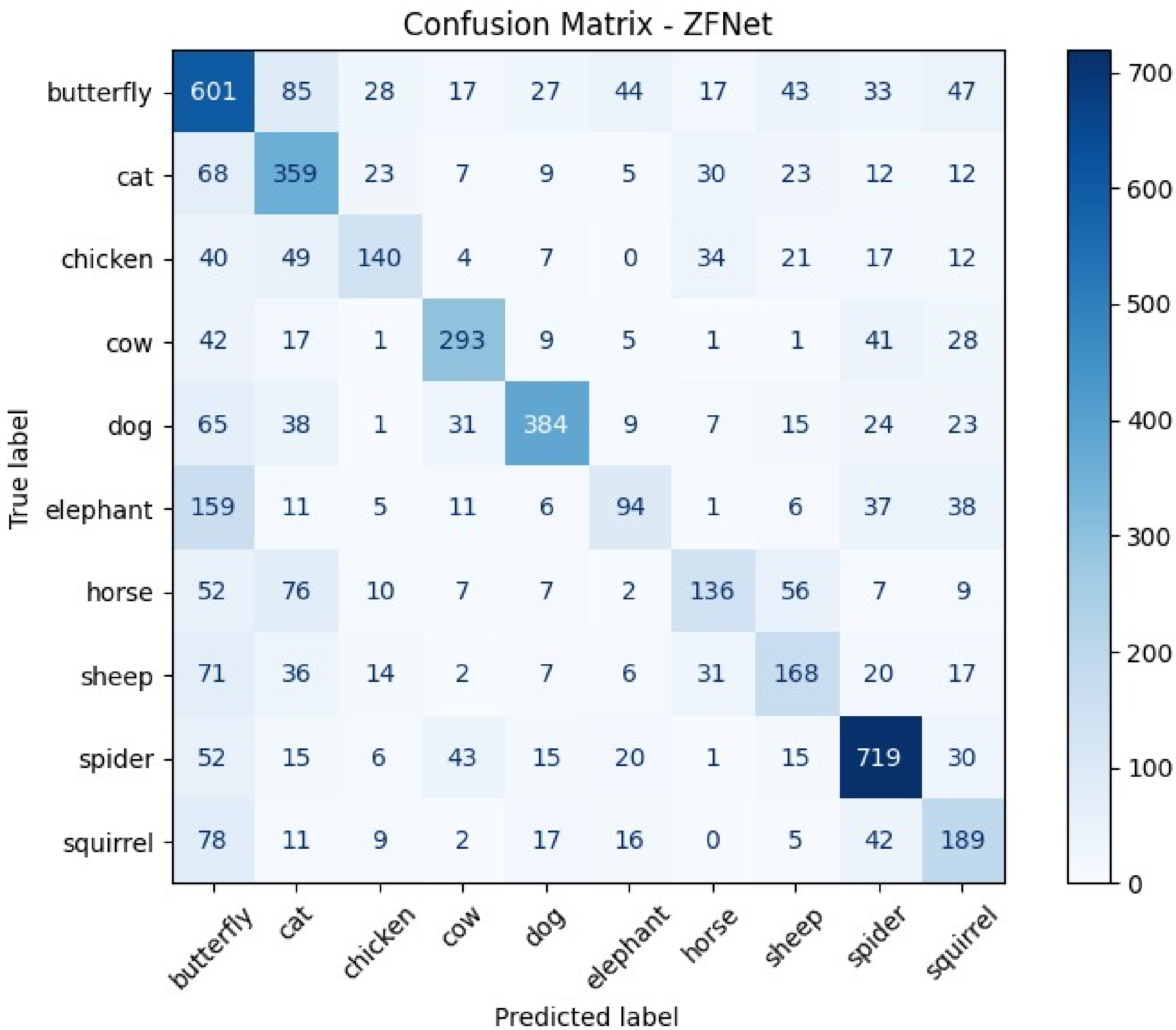
```
    plt.ylabel('Accuracy')
    plt.legend()

    # Loss
    plt.subplot(1, 2, 2)
    for name, result in results.items():
        loss = result['history']['train_loss']
        plt.plot(loss, label=f'{name}')
    plt.title('Training Loss over Epochs')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.suptitle("Training Performance Comparison", fontsize=18)
    plt.tight_layout()
    plt.show()

plot_training_history(results)
```

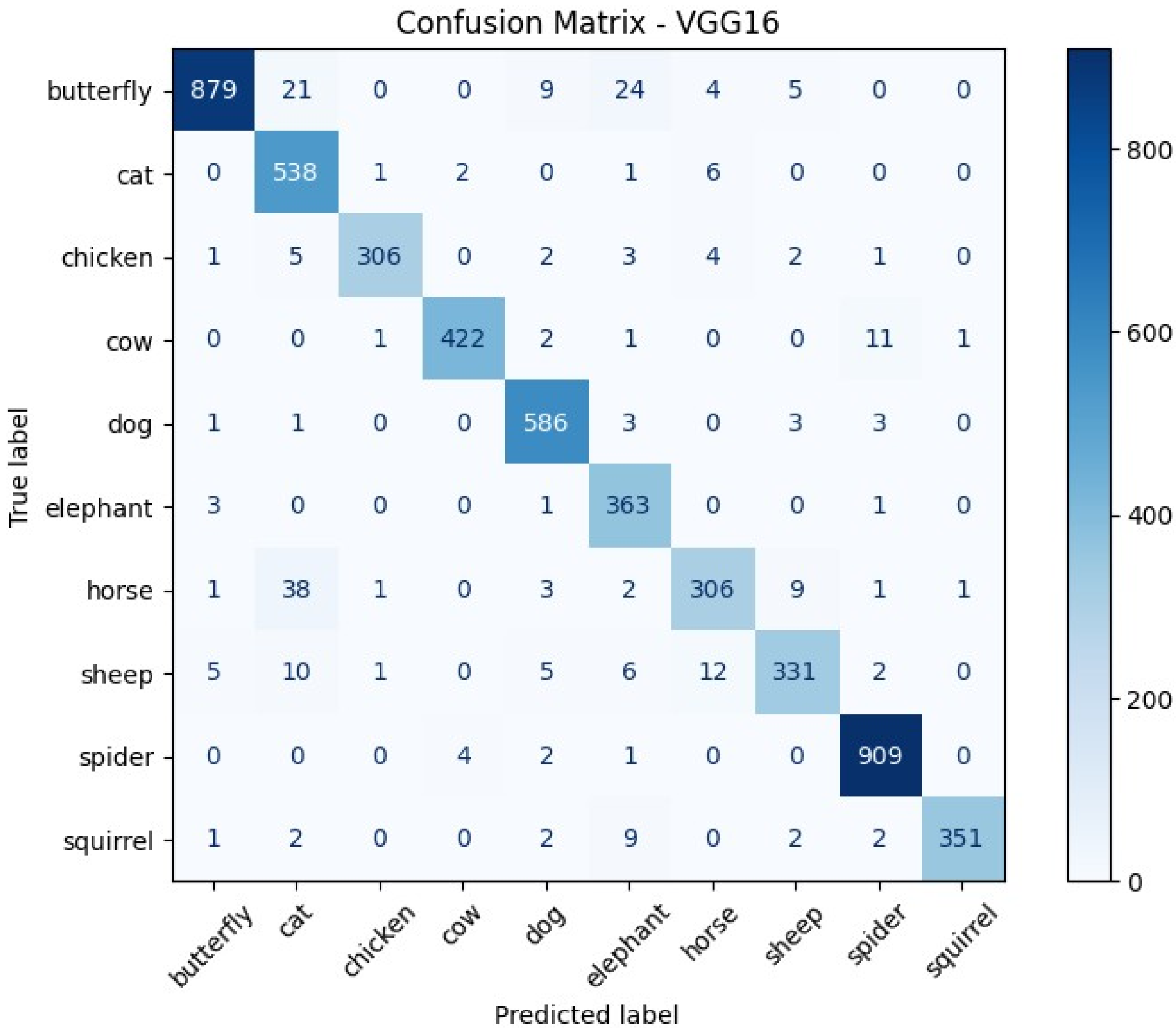


Plot Confusion Matrix for ZFNet, VGG16, and GoogLeNet

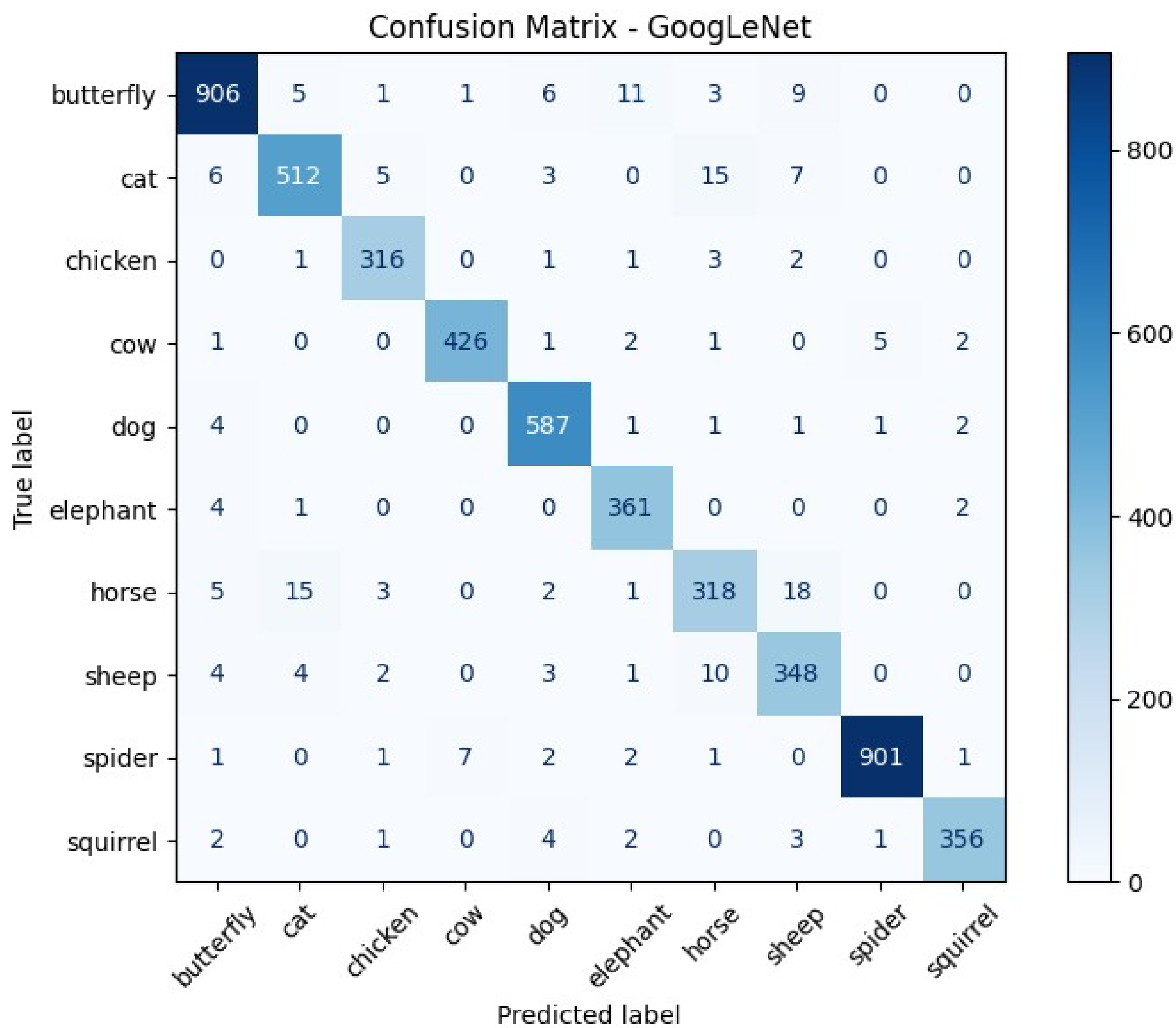
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

def plot_confusion_matrix(model, name):
    model.eval()
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for images, labels in val_loader:
            images = images.to(device)
            labels = labels.to(device)
            outputs = model(images)
            _, preds = torch.max(outputs, 1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())
```

```
    cm = confusion_matrix(all_labels, all_preds)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=class_names)
    fig, ax = plt.subplots(figsize=(8, 6))
    disp.plot(ax=ax, cmap='Blues', xticks_rotation=45)
    plt.title(f'Confusion Matrix - {name}')
    plt.tight_layout()
    plt.show()

# Plot confusion matrices
for model_name, result in results.items():
    plot_confusion_matrix(result['model'], model_name)
```





Confusion Matrix - VGG16
True label
Predicted label
butterfly
cat
chicken
cow
dog
elephant
horse
sheep
spider
squirrel
879 21 0 0 9 24 4 5 0 0
0 538 1 2 0 1 6 0 0 0
1 5 306 0 2 3 4 2 1 0
0 0 1 422 2 1 0 0 11 1
1 1 0 0 586 3 0 3 3 0
3 0 0 0 1 363 0 0 1 0
1 38 1 0 3 2 306 9 1 1
5 10 1 0 5 6 12 331 2 0
0 0 0 4 2 1 0 0 909 0
1 2 0 0 2 9 0 2 2 351
0
200
400
600
800



Evaluate Models and Generate Combined Precision, Recall, F1-Score, and Accuracy Table

```
class_names = [
    'butterfly', 'cat', 'chicken', 'cow', 'dog',
    'elephant', 'horse', 'sheep', 'spider', 'squirrel'
]
from sklearn.metrics import classification_report, accuracy_score
import pandas as pd

def evaluate_model(model, model_name):
    model.eval()
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
```

```
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    report = classification_report(all_labels, all_preds,
target_names=class_names, output_dict=True)
    accuracy = accuracy_score(all_labels, all_preds)

    return report, accuracy

# Evaluate all models
model_reports = {}
for model_name, result in results.items():
    print(f"Evaluating {model_name}...")
    report, accuracy = evaluate_model(result['model'], model_name)
    model_reports[model_name] = {'report': report, 'accuracy':
accuracy}

# Create combined DataFrame
metrics = ['precision', 'recall', 'f1-score']
rows = []

for class_name in class_names:
    row = {'Class': class_name}
    for model_name in model_reports:
        for metric in metrics:
            value = model_reports[model_name]['report'][class_name]
[metric]
            row[f"{model_name}_{metric}"] = round(value, 3)
    rows.append(row)

# Add overall accuracy
acc_row = {'Class': 'Overall Accuracy'}
for model_name in model_reports:
    acc = model_reports[model_name]['accuracy']
    acc_row[f"{model_name}_precision"] = acc
    acc_row[f"{model_name}_recall"] = acc
    acc_row[f"{model_name}_f1-score"] = acc
rows.append(acc_row)

# Convert to DataFrame and display
df = pd.DataFrame(rows)
pd.set_option("display.max_columns", None)
print(df)
```

```
Evaluating ZFNet...
Evaluating VGG16...
Evaluating GoogLeNet...
               Class  ZFNet_precision  ZFNet_recall  ZFNet_f1-score  \
```

```
0          butterfly         0.489000      0.638000        0.554000
1                cat         0.515000      0.655000        0.577000
2            chicken         0.591000      0.432000        0.499000
3                cow         0.703000      0.669000        0.685000
4                dog         0.787000      0.643000        0.708000
5           elephant         0.468000      0.255000        0.330000
6              horse         0.527000      0.376000        0.439000
7              sheep         0.476000      0.452000        0.463000
8             spider         0.755000      0.785000        0.770000
9           squirrel         0.467000      0.512000        0.488000
10  Overall Accuracy         0.588808      0.588808        0.588808

    VGG16_precision  VGG16_recall  VGG16_f1-score  GoogLeNet_precision
\
0          0.987000      0.933000        0.959000             0.971000

1          0.875000      0.982000        0.925000             0.952000

2          0.987000      0.944000        0.965000             0.960000

3          0.986000      0.963000        0.975000             0.982000

4          0.958000      0.982000        0.969000             0.964000

5          0.879000      0.986000        0.930000             0.945000

6          0.922000      0.845000        0.882000             0.903000

7          0.940000      0.890000        0.914000             0.897000

8          0.977000      0.992000        0.985000             0.992000

9          0.994000      0.951000        0.972000             0.981000

10         0.953209      0.953209        0.953209             0.960848


    GoogLeNet_recall  GoogLeNet_f1-score
0           0.962000            0.966000
1           0.934000            0.943000
2           0.975000            0.968000
3           0.973000            0.977000
4           0.983000            0.973000
5           0.981000            0.963000
6           0.878000            0.891000
7           0.935000            0.916000
8           0.984000            0.988000
9           0.965000            0.973000
10          0.960848            0.960848
```

```python
import matplotlib.pyplot as plt

# Extract final epoch values for each model
model_names = list(results.keys())
final_accuracies = [results[m]['history']['val_acc'][-1] for m in
model_names]
final_losses = [results[m]['history']['train_loss'][-1] for m in
model_names]

# Plot Validation Accuracy
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.bar(model_names, final_accuracies, color=['skyblue', 'salmon',
'lightgreen'])
plt.title("Final Validation Accuracy Comparison")
plt.ylabel("Accuracy")
plt.ylim(0, 1)  # accuracy ranges from 0 to 1
for i, v in enumerate(final_accuracies):
    plt.text(i, v + 0.01, f"{v:.2f}", ha='center', fontweight='bold')

# Plot Training Loss
plt.subplot(1, 2, 2)
plt.bar(model_names, final_losses, color=['skyblue', 'salmon',
'lightgreen'])
plt.title("Final Training Loss Comparison")
plt.ylabel("Loss")
for i, v in enumerate(final_losses):
    plt.text(i, v + 0.01, f"{v:.2f}", ha='center', fontweight='bold')

plt.tight_layout()
plt.show()
```