1. Describe the input and output for each model, hardware requirement, data statistic, learning curve, metrics (train text val), demo the result, finetuning technique, etc.

    a. **Notebook 1**: Model with CNN Architecture
        - **Input:** RGB images of size 32x32 pixels.
        - **Output:** Probabilities for 10 classes.
        - **Hardware Requirement:** GPU recommended for faster training.
        - **Data Statistic:**
            1. **Total:** CIFAR-10 dataset with 60,000 32x32 color images in 10 classes
            2. **Train:** 40,000 32x32 color images in 10 classes
            3. **Test:** 10,000 32x32 color images in 10 classes
            4. **Validate:** 10,000 32x32 color images in 10 classes
        - **Learning Curve:**
            1. **Loss:** Decreased by time and look like exponential decay.
            2. **Accuracy:** Increased with time until it was close to 0.5, and the rate of increase gradually decreased.
        - **Metrics:**
            1. Classification Accuracy
            2. Cross-Entropy Loss
            3. F1 Score
            4. Confusion Matrix
        - **Demo Result:** No demo result, only show the sample data with label in grid format and testing result.
        - **Fine-tuning Technique:** Stochastic Gradient Descent (SGD) with Cross-Entropy Loss
        - **Total Params:** 62,006
        - **Optimizer:** Stochastic Gradient Descent (SGD)
        - **Loss:** Cross-Entropy Loss
        - **Learning rate:** fixed 0.01
        - **Epoch:** 20

b. **Notebook 2**: Model with EfficientNet V2 S Architecture

- **Input:** RGB images of size 224x224 pixels.
- **Output:** Probabilities for 10 animal classes.
- **Hardware Requirement:** GPU strongly recommended due to model complexity.
- **Data Statistic:**
  1. **Total:** animal dataset with 2,000 224x224 color images in 10 classes (butterfly, cat, chicken, cow, dog, elephant, horse, sheep, spider, squirrel)
  2. **Train:** 1,400 224x224 color images in 10 classes
  3. **Test:** 300 224x224 color images in 10 classes
  4. **Validate:** 300 224x224 color images in 10 classes
- **Learning Curve:**
  1. **Loss:** Decreased by time and the rate of decrease gradually decreased.
  2. **Accuracy:** Increased with time until it was close to 1.0, and the rate of increase gradually decreased.
- **Metrics:**
  1. Classification Accuracy
  2. Cross-Entropy Loss
  3. F1 Score
  4. Confusion Matrix
- **Demo Result:** No demo result, only show the sample data with label in grid format and testing result.
- **Fine-tuning Technique:** Transfer learning with pre-trained EfficientNetV2s model by using SGD optimizer with Cross-Entropy Loss and Learning rate scheduler.
- **Total Params:** 20,190,298
- **Optimizer:** Stochastic Gradient Descent (SGD)
- **Loss:** Cross-Entropy Loss
- **Learning rate:** Step learning rate with start from 0.02, step_size=7 and gamma=0.5
- **Epoch:** 20

2. List key features for each function, including input and output. (cheat sheet)

a. Notebook 1:

– **Transformer**

```
transform = transforms.Compose( # transform is from torchvision (only for image)
    [transforms.ToTensor(), # image to tensor --> divide by 255
     transforms.Resize((32, 32))])
```

- \`transforms.Compose\`
- Input
    - list of transforms to compose.
- Output
    - Transformer that can transform in sequentially with transforms in the list.

– **Data loading**

```
trainvalset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
trainset, valset = torch.utils.data.random_split(trainvalset, [40000, 10000])

trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size, shuffle=True)
valloader = torch.utils.data.DataLoader(valset, batch_size=batch_size, shuffle=False)

testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size, shuffle=False)
```

- \`torchvision.datasets.CIFAR10\`, \`torch.utils.data.DataLoader\`
- Input
    - root (directory to save/download CIFAR-10 dataset)
    - train (True for training set, False for testing set)
    - download (True to download dataset if not available)
    - transform (data preprocessing and augmentation)
- Output
    - data loaders (function that can make accessing the data with transformer easier)

– **Model**

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

class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5) # 3 input channels, 6 output channels, 5*5 kernel size
        self.pool = nn.MaxPool2d(2, 2) # 2*2 kernel size, 2 strides
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(400, 120) # dense input 400 (16*5), output 120

        self.fc2 = nn.Linear(120, 84) # dense input 120, output 84
        self.fc3 = nn.Linear(84, 10) # dense input 84, output 10
        self.softmax = torch.nn.Softmax(dim=1) # perform softmax at dim[1] (batch,class)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x,start_dim=1) # flatten all dimensions (dim[1]) except batch (dim[0])
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        x = self.softmax(x)
        return x

net = CNN().to(device)
```

- \`class CNN(nn.Module)\`
- Input
    - None
- Output
    - CNN model with defined layers and activation

- o

– **Model Summary**

```python
from torchinfo import summary as summary_info
print(summary_info(net, input_size = (32, 3, 32, 32))) # (batchsize,channel,width,height)
net = net.to(device)
```

- `summary_info`
- Input:
  - o Model, input_size
- Output:
  - o Summary of model (Parameter in each layers and total params)

– **Training Loop**

```python
from sklearn.metrics import classification_report
from tqdm.notebook import tqdm


epochs = 20

history_train = {'loss':np.zeros(epochs), 'acc':np.zeros(epochs), 'f1-score':np.zeros(epochs)}
history_val = {'loss':np.zeros(epochs), 'acc':np.zeros(epochs), 'f1-score':np.zeros(epochs)}
min_val_loss = 1e10
PATH = './Animal10-efficientnetV2s.pth'

for epoch in range(epochs):  # loop over the dataset multiple times

    print(f'epoch {epoch + 1} \nTraining ...')
    net.train()
    y_predict = list()
    y_labels = list()
    training_loss = 0.0
    n = 0
    with torch.set_grad_enabled(True):
        for data in tqdm(trainloader):

            # get the inputs; data is a list of [inputs, labels]
            inputs, labels = data
            inputs = inputs.to(device)
            labels = labels.to(device)

            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            # aggregate statistics
            training_loss += loss.item()
            n+=1

            y_labels += list(labels.cpu().numpy())
            y_predict += list(outputs.argmax(dim=1).cpu().numpy())
    scheduler.step()

    # print statistics
    report = classification_report(y_labels, y_predict, digits = 4, output_dict = True)
    acc = report["accuracy"]
    f1 = report["weighted avg"]["f1-score"]
    support = report["weighted avg"]["support"]
    training_loss /= n
    print(f"training loss: {training_loss:.4}, acc: {acc*100:.4}%, f1-score: {f1*100:.4}%, support: {support}" )
    history_train['loss'][epoch] = training_loss
    history_train['acc'][epoch] = acc
    history_train['f1-score'][epoch] = f1

    print('validating ...')
    net.eval()

    optimizer.zero_grad()

    y_predict = list()
    y_labels = list()
    validation_loss = 0.0
    n = 0
    with torch.no_grad():
        for data in tqdm(valloader):
            inputs, labels = data
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = net(inputs)
            loss = criterion(outputs, labels)
            validation_loss += loss.item()

            y_labels += list(labels.cpu().numpy())
            y_predict += list(outputs.argmax(dim=1).cpu().numpy())
            n+=1

    # print statistics
    report = classification_report(y_labels, y_predict, digits = 4, output_dict = True)
    acc = report["accuracy"]
    f1 = report["weighted avg"]["f1-score"]
    support = report["weighted avg"]["support"]
    validation_loss /= n
    print(f"validation loss: {validation_loss:.4}, acc: {acc*100:.4}%, f1-score: {f1*100:.4}%, support: {support}" )
    history_val['loss'][epoch] = validation_loss
    history_val['acc'][epoch] = acc
    history_val['f1-score'][epoch] = f1

    #save min validation loss
    if validation_loss < min_val_loss:
        torch.save(net.state_dict(), PATH)
        min_val_loss = validation_loss

print('Finished Training')
```

- Training loop using `torch.optim.SGD`, `torch.nn.CrossEntropyLoss`
- Input
  - o trainloader, valloader, CNN model, CrossEntropyLoss, SGD optimizer, number of epochs
- Output
  - o Process bar
  - o Trained CNN model, training, and validation history

– **Learning Visualization**

```
fig, axs = plt.subplots(3, figsize= (6,10))
# loss
axs[0].plot(history_train['loss'], label = 'training')
axs[0].plot(history_val['loss'], label = 'validation')
axs[0].set_title("loss")
axs[0].legend()
# acc
axs[1].plot(history_train['acc'], label = 'training')
axs[1].plot(history_val['acc'], label = 'validation')
axs[1].set_title("acc")
axs[1].legend()
# f1-score
axs[2].plot(history_train['f1-score'], label = 'training')
axs[2].plot(history_val['f1-score'], label = 'validation')
axs[2].set_title("f1-score")
axs[2].legend()
plt.show()
```

- `plot`, `show`
- Input
  - history_train, history_val
- Output
  - Graph that display loss, acc, f1-score

– **Evaluation Metrics**

```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay

print('testing ...')
y_predict = list()
y_labels = list()
test_loss = 0.0
n = 0
with torch.no_grad():
    for data in tqdm(testloader):
        inputs, labels = data
        inputs = inputs.to(device)
        labels = labels.to(device)

        outputs = net(inputs)
        loss = criterion(outputs, labels)
        test_loss += loss.item()

        y_labels += list(labels.cpu().numpy())
        y_predict += list(outputs.argmax(dim=1).cpu().numpy())
        n+=1

    # print statistics
    test_loss /= n
    print(f"testing loss: {test_loss:.4}" )

    report = classification_report(y_labels, y_predict, digits = 4)
    M = confusion_matrix(y_labels, y_predict)
    print(report)
    disp = ConfusionMatrixDisplay(confusion_matrix=M)
```

- `classification_report`, `confusion_matrix`
- Input
  - Trained CNN model, testloader
- Output
  - Classification report, confusion matrix