1. Upload a dataset (`final_dataset1000.jsonl`) and a model (`model_v2.py`, `model_v2_1.py`, `model_v3.py`, `model_v3_1.py`)
2. In `# DATASET PATH` cell, fill `<your_dataset_path>` in `df = pd.read_json('<your_dataset_path>', lines=True)` with the path of the uploaded dataset.
3. In `# import model from .py file here` cell import `<your_model.py>` by `from <your_model> import <the class in your_model.py>`
4. In `# Load model` cell, let `model = <the class in your_model.py>` with proper parameters (model_name and num_classes which should be already provided). Name `model_id` with v2, v2_1, v3 or v3_1 based on the imported .py file.
5. After running all cells, you should see graphs printed. Save the graphs.
6. You should also see other files saved in the files section along with your uploaded dataset file and model file. Save those files: `.json` files and a `.pth` file of the best performed model.

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.nn.init as nn_init
from torch.optim import AdamW
from torch.utils.data import Dataset, TensorDataset, DataLoader
from torch.nn.utils.rnn import pad_sequence
from transformers import AutoModel, AutoTokenizer
import pandas as pd
import json
from sklearn.model_selection import train_test_split


model_name = "xlm-roberta-base"
tokenizer = AutoTokenizer.from_pretrained(model_name)
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100%  25.0/25.0 [00:00<00:00, 3.18kB/s]
config.json: 100%  615/615 [00:00<00:00, 78.3kB/s]
sentencepiece.bpe.model: 100%  5.07M/5.07M [00:00<00:00, 5.97MB/s]
tokenizer.json: 100%  9.10M/9.10M [00:00<00:00, 10.4MB/s]
```

```
# DATASET PATH
df = pd.read_json('/content/final_dataset1000.jsonl', lines=True)
df_combined = df
print(len(df_combined))
```

```
13983
```

```
# Czech uses Latin characters with diacritics (Latin Extended-A block)
czech_chars = [chr(i) for i in range(0x0100, 0x0180) if chr(i).isprintable()]  # Czech (Latin Extended-A)

# Greek alphabet (Greek and Coptic Unicode block)
greek_chars = [chr(i) for i in range(0x0370, 0x0400) if chr(i).isprintable()]  # Greek

# Hebrew alphabet (Hebrew Unicode block)
hebrew_chars = [chr(i) for i in range(0x0590, 0x0600) if chr(i).isprintable()]  # Hebrew

# Russian alphabet (Cyrillic Unicode block)
russian_chars = [chr(i) for i in range(0x0400, 0x0500) if chr(i).isprintable()]  # Russian (Cyrillic)

# Arabic alphabet (Arabic Unicode block)
arabic_chars = [chr(i) for i in range(0x0600, 0x0700) if chr(i).isprintable()]  # Arabic

# Korean characters (Hangul Syllables Unicode block)
korean_chars = [chr(i) for i in range(0xAC00, 0xD7A4) if chr(i).isprintable()]  # Korean (Hangul syllables)

# Macedonian alphabet (using the Cyrillic Supplement block for uniqueness)
macedonian_chars = [chr(i) for i in range(0x0500, 0x0530) if chr(i).isprintable()]  # Macedonian (Cyrillic Supplement)

# Thai characters (Thai Unicode block)
thai_chars = [chr(i) for i in range(0x0E00, 0x0E80) if chr(i).isprintable()]  # Thai

# Hindi characters (Devanagari Unicode block)
hindi_chars = [chr(i) for i in range(0x0900, 0x0980) if chr(i).isprintable()]  # Hindi (Devanagari)

# Bengali characters (Bengali Unicode block)
bengali_chars = [chr(i) for i in range(0x0980, 0x0A00) if chr(i).isprintable()]  # Bengali
```

```
print("Czech:", "".join(czech_chars))
print("Greek:", "".join(greek_chars))
print("Hebrew:", "".join(hebrew_chars))
print("Russian:", "".join(russian_chars))
print("Arabic:", "".join(arabic_chars))
print("Korean:", "".join(korean_chars[:50]))    # output only first 50 for brevity
print("Macedonian:", "".join(macedonian_chars))
print("Thai:", "".join(thai_chars))
print("Hindi:", "".join(hindi_chars))
print("Bengali:", "".join(bengali_chars))


latin_chars = [chr(i) for i in range(0x0020, 0x007B) if chr(i).isprintable()]  # Basic Latin (A-Z, a-z)
chinese_chars = [chr(i) for i in range(0x4E00, 0x9FFF) if chr(i).isprintable()]  # Common Chinese characters

# French uses Latin characters with accents
french_chars = [chr(i) for i in range(0x00C0, 0x0100) if chr(i).isprintable()]  # À-ÿ (includes é, è, ç, etc.)

# Japanese includes Hiragana, Katakana
hiragana_chars = [chr(i) for i in range(0x3041, 0x30A0) if chr(i).isprintable()]  # Hiragana
katakana_chars = [chr(i) for i in range(0x30A1, 0x3100) if chr(i).isprintable()]  # Katakana



# Combine all characters and verify uniqueness
unique_chars = set("".join(
    czech_chars + greek_chars + hebrew_chars + russian_chars +
    arabic_chars + korean_chars + macedonian_chars + thai_chars +
    hindi_chars + bengali_chars + latin_chars + chinese_chars +
    french_chars + hiragana_chars + katakana_chars
))
print(len(unique_chars))
print(len(unique_chars) == len(czech_chars) + len(greek_chars) + len(hebrew_chars) +
      len(russian_chars) + len(arabic_chars) + len(korean_chars) + len(macedonian_chars) +
      len(thai_chars) + len(hindi_chars) + len(bengali_chars) + len(latin_chars) +
      len(chinese_chars) + len(french_chars) + len(hiragana_chars) + len(katakana_chars))
```

```
Czech: ĀāĂăĄąĆćĈĉĊċČčĎďĐđĒēĔĕĖėĘęĚěĜĝĞğĠġĢģĤĥĦħĨĩĪīĬĭĮįİıĲĳĴĵĶķĸĹĺĻļĽľĿŀŁłŃńŅņŇňŉŊŋŌōŎŏŐőŒœŔŕŖŗŘřŚśŜŝŞşŠšŢţŤťŦŧŨũŪūŬŭŮůŰűŲųŴŵŶŷ
Greek: ͰͱͲͳʹ͵Ͷͷͺͻͼͽ;Ϳ΄΅Ά·ΈΉΊΌΎΏΐΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩΪΫάέήίΰαβγδεζηθικλμνξοπρςστυφχψωϊϋόύώ
Hebrew: ׇ׃ׂאבגדהוזחטיךכלםמןנסעףפץצקרשתׯװױײ׳״
Russian: ЀЁЂЃЄЅІЇЈЉЊЋЌЍЎЏАБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдежзийклмнопрстуфхцчш
Arabic: ٹٺٻټٽپٿڀځڂڃڄڅچڇڈډڊڋڌڍڎڏڐڑڒړڔڕږڗژڙښڛڜڝڞڟڠڡڢڣڤڥڦڧڨکڪګڬڭڮگڰڱڲڳڴڵڶڷڸڹںڻڼڽھڿۀہۂۃۄۅۆۇۈۉۊۋیۍێۏېۑےۓ۔ە۝۞ۮۯ۰۱۲۳۴۵۶۷۸۹ۺۻۼ۽۾ۿ
Korean: 가각갂갃간갅갆갇갈갉갊갋갌갍갎갏감갑값갓갔강갖갗갘같갚갛개객갞갟갠갡갢갣갤갥갦갧갨갩갪갫갬갭갮갯갰갱
Macedonian: ԀԁԂԃԄԅԆԇԈԉԊԋԌԍԎԏԐԑԒԓԔԕԖԗԘԙԚԛԜԝԞԟԠԡԢԣԤԥԦԧԨԩԪԫԬԭԮԯ
Thai: กขฃคฅฆงจฉชซฌญฎฏฐฑฒณดตถทธนบปผฝพฟภมยรฤลฦวศษสหฬอฮฯะัาำิีึืฺุู฿เแโใไๅๆ็่้๊๋์ํ๎๏๐๑๒๓๔๕๖๗๘๙๚๛
Hindi: ऀँंःऄअआइईउऊऋऌऍऎएऐऑऒओऔकखगघङचछजझञटठडढणतथदधनऩपफबभमयरऱलळऴवशषसहऺऻ़ऽािीुूृॄॅॆेैॉॊोौ्ॎॏॐ॒॑॓॔ॕॖॗक़ख़ग़ज़ड़ढ़फ़य़ॠॡॢॣ।॥०१२३४५६७८९॰ॱॲॳॴॵॶॷॸॹॺॻॼॽॾॿ
Bengali: ঀঁংঃঅআইঈউঊঋঌএঐওঔকখগঘঙচছজঝঞটঠডঢণতথদধনপফবভমযরলশষসহ়ঽািীুূৃৄেৈোৌ্ৎৗড়ঢ়য়ৠৡৢৣ০১২৩৪৫৬৭৮৯ৰৱ৲৳৴৵৶৷৸৹৺৻ৼ৽৾
33720
True
```

```
# manually extra
unique_chars.add("¦")
unique_chars.add('\u3000')
unique_chars.add('、')
unique_chars.add('。')
unique_chars.add('？')
unique_chars.add('！')
unique_chars.add('「')
unique_chars.add('」')
unique_chars.add('『')
unique_chars.add('』')
unique_chars.add('《')
unique_chars.add('》')
unique_chars.add('ー')
unique_chars.add('０')
unique_chars.add('ゞ')
unique_chars.add('♪')
unique_chars.add('¿')
unique_chars.add('\x87')
unique_chars.add('\u200d')
print(len(unique_chars))
```

```
33739
```

```
for text in df_combined['text']:
    unique_chars.update(text)
```

```
# Define large Unicode character mapping
char_to_idx = {char: idx for idx, char in enumerate(unique_chars)}
idx_to_char = {idx: char for char, idx in char_to_idx.items()}
num_classes = len(unique_chars)
print(num_classes)
```

```
33884
```

```
# Function to split the text in the middle and extract the input and label
def split_text(row):
    text = row['text']
    mid_point = len(text) // 2

    input_text = text[:mid_point]
    label_text = text[mid_point:]

    label = label_text[0] if len(label_text) > 0 else ""

    return pd.Series([input_text, label])

# Apply the function and assign to new columns
df_combined[['input', 'label']] = df_combined.apply(split_text, axis=1)

def tokenize_text(text):
    if tokenizer.pad_token is None:
        tokenizer.add_special_tokens({'pad_token': '[PAD]'})
    return tokenizer(text, padding=True, truncation=True, max_length=50, return_tensors="pt")
def label_to_idx(label):
    return char_to_idx[label]

# Apply the tokenizer to the 'text' column and store the tokenized results
df_combined['tokenized_input'] = df_combined['input'].apply(lambda x: tokenize_text(x))
df_combined['label_idx'] = df_combined['label'].apply(lambda x: label_to_idx(x))
df_combined['input_ids'] = df_combined['tokenized_input'].apply(lambda x: x['input_ids'].squeeze(0))
df_combined['attention_mask'] = df_combined['tokenized_input'].apply(lambda x: x['attention_mask'].squeeze(0))
df_combined.drop(columns=['tokenized_input'], inplace=True)
df_combined = df_combined.reset_index(drop=True)
print(df_combined.head())
print(df_combined.iloc[0]['input_ids'])
```

```
  language                                               text  \
0   Arabic                                          .حسنًا
1   Arabic       أعرف ان كتابة المخططات ليس ممتعاً كالجراحات
2   Arabic                          هاك. إرتدِ هذا, هلا فعلت؟
3   Arabic  محاولاً الحصول على فكرة بينما تخرج زوجتي من ال...
4   Arabic                                               !

                      input label  label_idx  \
0                      22727      ن     حس
1      14416      ت     أعرف ان كتابة المخططا
2              15182      ذ     هاك. إرتدِ ه
3  19794      ي     محاولاً الحصول على فكرة ب
4                          !       9658

                                        input_ids  \
0  [tensor(0), tensor(12106), tensor(3247), tenso...
1  [tensor(0), tensor(1333), tensor(48753), tenso...
2  [tensor(0), tensor(917), tensor(972), tensor(5...
3  [tensor(0), tensor(63050), tensor(40286), tens...
4                          [tensor(0), tensor(2)]

                                   attention_mask
0  [tensor(1), tensor(1), tensor(1), tensor(1), t...
1  [tensor(1), tensor(1), tensor(1), tensor(1), t...
2  [tensor(1), tensor(1), tensor(1), tensor(1), t...
3  [tensor(1), tensor(1), tensor(1), tensor(1), t...
4                          [tensor(1), tensor(1)]
tensor([    0, 12106,  3247,   926,     2])
```

```
train_df, val_df = train_test_split(df_combined, test_size=0.2, random_state=42)
print(len(train_df))
print(len(val_df))
```

```
11186
2797
```

```
"""
Usage:
dataset = TextDataset(df)
dataloader = DataLoader(dataset, batch_size=<x>, shuffle=True)
"""

class TextDataset(Dataset):
```

```
        def __init__(self, df):
            self.input_ids = df['input_ids'].tolist()
            self.attention_mask = df['attention_mask'].tolist()
            self.labels = df['label_idx'].tolist()
            self.language = df['language'].tolist()

        def __len__(self):
            return len(self.labels)

        def __getitem__(self, idx):
            input_ids = self.input_ids[idx]
            attention_mask = self.attention_mask[idx]
            label = self.labels[idx]
            language = self.language[idx]

            return {
                'input_ids': input_ids,
                'attention_mask': attention_mask,
                'labels': torch.tensor(label),
                'language': language
            }


# import model from .py file here
from model_v2 import UnicodeClassifier_v2


class UnicodeClassifier(nn.Module):
    def __init__(self, model_name, num_classes, freeze_encoder = True):
        super(UnicodeClassifier, self).__init__()
        self.encoder = AutoModel.from_pretrained(model_name)
        self.fc1 = nn.Linear(self.encoder.config.hidden_size, 512)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(512, 256)
        self.relu2 = nn.ReLU()
        self.fc = nn.Linear(256, num_classes)  # Large Unicode classification head

        if freeze_encoder:
            for param in self.encoder.parameters():
                param.requires_grad = False

        self._initialize_weights()

    def _initialize_weights(self):
        for m in self.fc.modules():
            if isinstance(m, nn.BatchNorm1d):
                m.weight.data.fill_(1)  # gamma to 1
                m.bias.data.zero_()  # beta to 0
            elif isinstance(m, nn.Linear):
                nn_init.xavier_uniform_(m.weight)  # Xavier initialization
                if m.bias is not None:
                    m.bias.data.zero_()  # bias to 0

    def forward(self, input_ids: torch.tensor, attention_mask: torch.tensor):
        outputs = self.encoder(input_ids=input_ids, attention_mask=attention_mask)
        outputs = self.relu1(self.fc1(outputs.last_hidden_state[:, 0, :]))
        outputs = self.relu2(self.fc2(outputs))
        logits = self.fc(outputs)  # CLS token output
        return logits


class TopKLoss(nn.Module):
    def __init__(self):
        super(TopKLoss, self).__init__()
        self.ce_loss = nn.CrossEntropyLoss()

    def forward(self, logits, target):
        loss = self.ce_loss(logits, target)

        return loss


def collate_fn(batch):
    """
    Custom collate function to pad sequences within a batch.
    """
    input_ids = [item['input_ids'] for item in batch]
    attention_mask = [item['attention_mask'] for item in batch]
    labels = torch.tensor([item['labels'] for item in batch])
    languages = [item['language'] for item in batch]

    # Pad input_ids and attention_mask
    input_ids = pad_sequence(input_ids, batch_first=True, padding_value=tokenizer.pad_token_id)
```

```
        attention_mask = pad_sequence(attention_mask, batch_first=True, padding_value=0)

        return {
            'input_ids': input_ids,
            'attention_mask': attention_mask,
            'labels': labels,
            'language': languages  # Assuming you need language information
        }


def evaluate(
        model: nn.Module,
        df_val: pd.DataFrame,
        collate_fn,
        eval_batch_size: int = 128,
        criterion: nn.Module = TopKLoss(),
        top_k: int = 3,
        device: str = "cpu",
):
        """
        Evaluate the model's loss on the validation set, also accuracy
        """
        dataset = TextDataset(df_val)
        val_dataloader = DataLoader(dataset, batch_size=eval_batch_size, collate_fn=collate_fn, shuffle=False)

        model.eval()
        model.to(device)

        val_loss = 0.0
        val_correct, val_samples = 0, 0

        with torch.no_grad():
            for batch in val_dataloader:
                input_ids_batch = batch['input_ids'].to(device)
                attention_mask_batch = batch['attention_mask'].to(device)
                y_batch = batch['labels'].to(device)

                y_batch_pred = model(input_ids_batch, attention_mask_batch)
                batch_loss = criterion(y_batch_pred, y_batch)

                top_k_probs, top_k_indices = torch.topk(y_batch_pred, k=top_k, dim=-1)
                batch_preds = [([idx.item() for idx in i]) for i in top_k_indices.cpu()]

                val_loss += batch_loss.item()
                batch_preds_tensor = torch.tensor(batch_preds, device=device)
                correct = (batch_preds_tensor == y_batch.view(-1, 1)).any(dim=1).float()
                val_correct += correct.sum().item()
                val_samples += len(batch_preds)

        val_loss /= len(val_dataloader)
        acc = float(val_correct) / val_samples

        return {"val_loss": val_loss, "val_acc": acc}




def train(
        model: nn.Module,
        df_train: pd.DataFrame,
        df_val: pd.DataFrame,
        collate_fn,
        lr: float = 1e-3,
        criterion: nn.Module = TopKLoss(),
        batch_size: int = 32,
        eval_batch_size: int = 128,
        num_epochs: int = 10,
        top_k: int = 3,
        device: str = "cpu",
        verbose: bool = True,
):
        """
        Run training loop for `n_epochs` epochs.
        """
        dataset = TextDataset(df_train)
        train_dataloader = DataLoader(
            dataset,
            batch_size=batch_size,
            collate_fn=collate_fn,
            shuffle=True,
        )

        model.to(device)
```

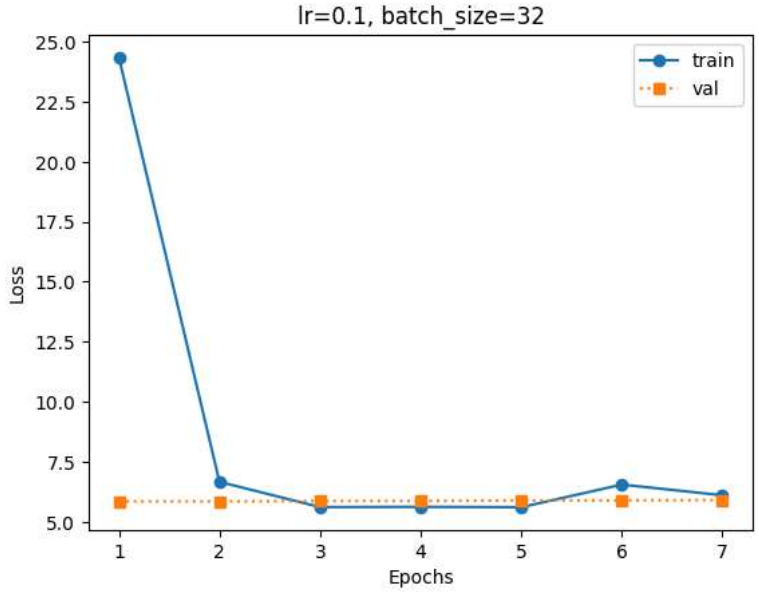
```
    optimizer = AdamW(model.parameters(), lr=lr)

    train_metrics = []
    val_metrics = []
    train_correct, train_samples = 0, 0

    for epoch in range(num_epochs):
        model.train()
        train_epoch_loss = 0.0
        for batch in train_dataloader:
            input_ids_batch = batch['input_ids'].to(device)
            attention_mask_batch = batch['attention_mask'].to(device)
            y_batch = batch['labels'].to(device)

            optimizer.zero_grad()
            y_batch_pred = model(input_ids_batch, attention_mask_batch)
            batch_loss = criterion(y_batch_pred, y_batch)
            batch_loss.backward()
            optimizer.step()

            top_k_probs, top_k_indices = torch.topk(y_batch_pred, k=top_k, dim=-1)
            batch_preds = [[idx.item() for idx in i] for i in top_k_indices.cpu()]

            train_epoch_loss += batch_loss.item()
            batch_preds_tensor = torch.tensor(batch_preds, device=device)
            correct = (batch_preds_tensor == y_batch.view(-1, 1)).any(dim=1).float()
            train_correct += correct.sum().item()
            train_samples += len(batch_preds)

        # train loss
        train_epoch_loss /= len(train_dataloader)

        # train acc
        train_acc = float(train_correct) / train_samples

        train_metrics.append({"train_loss": train_epoch_loss, "train_acc": train_acc})

        # val loss and acc
        eval_metrics = evaluate(model, df_val, collate_fn, eval_batch_size, criterion, top_k, device)
        val_metrics.append(eval_metrics)

        if verbose:
            print("Epoch: %d, Train Loss: %.4f, Train Acc: %.4f, Val Loss: %.4f, Val Acc: %.4f" % (epoch+1, train_epoch_

    return train_metrics, val_metrics


# Load model
model = UnicodeClassifier_v2(model_name, num_classes)
model_id = "v2" # v2, v2_1, v3, v3_1
```

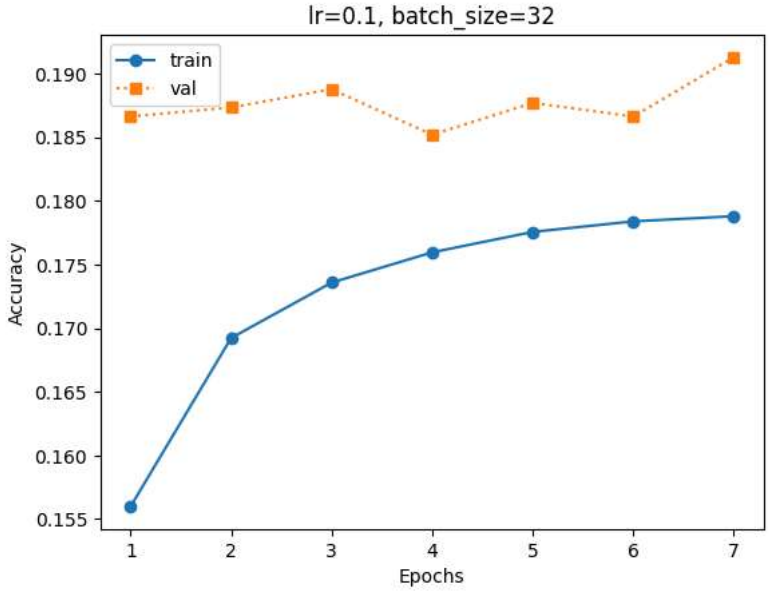
```
model.safetensors: 100%                                  1.12G/1.12G [00:05<00:00, 168MB/s]
```

```
# hyper parameters
lrs = [1e-1, 1e-3, 1e-5]

criterion = TopKLoss() # a new one?
batch_sizes = [32, 64, 128]
eval_batch_size = 128
num_epochs = 7
top_k = 3
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")


import matplotlib.pyplot as plt

def plot_metrics(train_metrics, val_metrics, setup):
    train_loss = [m['train_loss'] for m in train_metrics]
    train_acc = [m['train_acc'] for m in train_metrics]
    val_loss = [m['val_loss'] for m in val_metrics]
    val_acc = [m['val_acc'] for m in val_metrics]
    epochs = range(1, len(train_loss) + 1)

    def draw_plot(x, y1, y2, title, setup):
        plt.plot(x, y1, label = 'train', marker='o', linestyle='-')
        plt.plot(x, y2, label = 'val', marker='s', linestyle=':')
        plt.xlabel('Epochs')
        plt.ylabel(f'{title}')
        plt.title(f'{setup}')
        plt.legend()
        plt.show()
```

```
    draw_plot(epochs, train_loss, val_loss, 'Loss', setup)
    draw_plot(epochs, train_acc, val_acc, 'Accuracy', setup)


# DON'T forget to record the model name!!! => for a specific model with diff hyper parameters
for lr in lrs:
    for batch_size in batch_sizes:
        setup = f'lr={lr}, batch_size={batch_size}'
        train_metrics, val_metrics = train(model, train_df, val_df, collate_fn, lr, criterion, batch_size, eval_batch_size, num_epochs, top
        plot_metrics(train_metrics, val_metrics, setup)
        torch.save(model.state_dict(), f'{model_id}{setup}.pth')
        with open(f"{model_id}{setup}.json", "w") as f:
            json.dump(train_metrics, f, indent=4)
```

```
Epoch: 1, Train Loss: 24.3433, Train Acc: 0.1560, Val Loss: 5.8505, Val Acc: 0.1866
Epoch: 2, Train Loss: 6.6609, Train Acc: 0.1692, Val Loss: 5.8472, Val Acc: 0.1873
Epoch: 3, Train Loss: 5.6097, Train Acc: 0.1736, Val Loss: 5.8634, Val Acc: 0.1888
Epoch: 4, Train Loss: 5.6201, Train Acc: 0.1760, Val Loss: 5.8642, Val Acc: 0.1852
Epoch: 5, Train Loss: 5.6063, Train Acc: 0.1776, Val Loss: 5.8889, Val Acc: 0.1877
Epoch: 6, Train Loss: 6.5486, Train Acc: 0.1784, Val Loss: 5.8879, Val Acc: 0.1866
Epoch: 7, Train Loss: 6.1040, Train Acc: 0.1788, Val Loss: 5.9060, Val Acc: 0.1913
```






```
Epoch: 1, Train Loss: 17.1531, Train Acc: 0.1758, Val Loss: 6.0810, Val Acc: 0.1791
Epoch: 2, Train Loss: 6.4837, Train Acc: 0.1781, Val Loss: 6.0311, Val Acc: 0.1873
Epoch: 3, Train Loss: 7.4659, Train Acc: 0.1788, Val Loss: 6.0566, Val Acc: 0.1831
Epoch: 4, Train Loss: 5.7546, Train Acc: 0.1802, Val Loss: 6.0557, Val Acc: 0.1906
Epoch: 5, Train Loss: 8.2939, Train Acc: 0.1793, Val Loss: 6.0608, Val Acc: 0.1888
Epoch: 6, Train Loss: 5.8219, Train Acc: 0.1798, Val Loss: 6.0655, Val Acc: 0.1852
Epoch: 7, Train Loss: 7.5118, Train Acc: 0.1803, Val Loss: 6.0714, Val Acc: 0.1866
```