```python
model_name = 'bert-base-multilingual-uncased'
num_fold = 5
lr = 1e-5
max_len = 256
num_epoch = 5
batch_size = 32
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
tokenizer = AutoTokenizer.from_pretrained(model_name, do_lower_case=True)


kf = StratifiedKFold(n_splits=num_fold, shuffle=True, random_state=1016)
df = df_train
y = df['label']
fold_list = list(kf.split(df, y))
train_df_list = []
val_df_lsit = []
for i, fold in enumerate(fold_list):
    df_train = df[df.index.isin(fold[0])]
    df_val = df[df.index.isin(fold[1])]

    train_df_list.append(df_train)
val_df_lsit.append(df_val)


class CompDataset(Dataset):
    def __init__(self, df):
        self.data = df

    def __getitem__(self, index):
        sentence1 = self.data.loc[index, 'premise']
        sentence2 = self.data.loc[index, 'hypothesis']
        encode_dict = tokenizer.encode_plus(
            sentence1, sentence2,
            add_special_tokens=True,
            max_length=max_len,
            pad_to_max_length=True,
            return_attention_mask=True,
            return_tensors='pt',
            truncation=True,
        )
        input_ids = encode_dict['input_ids'][0]
        attention_mask = encode_dict['attention_mask'][0]
        token_type_ids = encode_dict['token_type_ids'][0]
        target = torch.tensor(self.data.loc[index, 'label'])

        sample = (input_ids, attention_mask, token_type_ids, target)
        return sample

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


class TestDataset(Dataset):
    def __init__(self, df):
        self.data = df

    def __getitem__(self, index):
        sentence1 = self.data.loc[index, 'premise']
        sentence2 = self.data.loc[index, 'hypothesis']
        encode_dict = tokenizer.encode_plus(
            sentence1, sentence2,
            add_special_tokens=True,
            max_length=max_len,
            pad_to_max_length=True,
            return_attention_mask=True,
            return_tensors='pt',
            truncation=True
        )
        input_ids = encode_dict['input_ids'][0]
        attention_mask = encode_dict['attention_mask'][0]
        token_type_ids = encode_dict['token_type_ids'][0]
        sample = (input_ids, attention_mask, token_type_ids)
        return sample

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


model = BertForSequenceClassification.from_pretrained(
    model_name,
    num_labels=3,
    output_attentions=False,
    output_hidden_states=False
)
model.to(device)
```

```python
fold_val_acc_list = []
for i in range(0, num_fold):
    fold_val_acc_list.append([])

for epoch in range(0, num_epoch):
    print("\nNum folds used for training:", 3)
    print('======== Epoch {:} / {:} ========'.format(epoch + 1, num_epoch))
    epoch_acc_scores_list = []
    for fold_index in range(0, 3):
        print('\n== Fold Model', fold_index)
        if epoch == 0:
            model = BertForSequenceClassification.from_pretrained(
                model_name,
                num_labels=3,
                output_attentions=False,
                output_hidden_states=False
            )
            model.to(device)
            optimizer = optim.AdamW(
                model.parameters(),
                lr=lr,
                eps=1e-8
            )
        else:
            path_model = 'model_' + str(fold_index) + '.bin'
            # model =AutoModel.from_pretrained(path_model)
            model.load_state_dict(torch.load(path_model))
            model.to(device)

        df_train = train_df_list[fold_index]
        df_val = val_df_lsit[fold_index]

        # Reset the indices or the dataloader won't work.
        df_train = df_train.reset_index(drop=True)
        df_val = df_val.reset_index(drop=True)

        # Create the dataloaders
        train_data = CompDataset(df_train)
        val_data = CompDataset(df_val)

        train_loader = torch.utils.data.DataLoader(train_data,
                                                   batch_size=batch_size,
                                                   shuffle=True,
                                                   )
```

```python
val_loader = torch.utils.data.DataLoader(val_data,
                                         batch_size=batch_size,
                                         shuffle=True,
                                         )

stacked_val_labels = []
targets_list = []

model.train()
print('Training...')
torch.set_grad_enabled(True)
total_train_loss = 0
for i, batch in enumerate(train_loader):
    train_status = 'Batch ' + str(i + 1) + ' of ' + str(len(train_loader))
    print(train_status, end='\r')
    b_input_ids = batch[0].to(device)
    b_attention_mask = batch[1].to(device)
    b_token_type_ids = batch[2].to(device)
    b_labels = batch[3].to(device)

    outputs = model(b_input_ids,
                    token_type_ids=b_token_type_ids,
                    attention_mask=b_attention_mask,
                    labels=b_labels
                    )
    loss = outputs[0]
    preds = outputs[1]
    total_train_loss = total_train_loss + loss.item()
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
print('Train loss:', total_train_loss)

model.eval()
print('\nValidation...')
total_val_loss = 0
for j, val_batch in enumerate(val_loader):
    val_status = 'Batch ' + str(j + 1) + ' of ' + str(len(val_loader))
    print(val_status, end='\r')
    b_input_ids = val_batch[0].to(device)
    b_attention_mask = val_batch[1].to(device)
    b_token_type_ids = val_batch[2].to(device)
    b_labels = val_batch[3].to(device)
```

```python
        outputs = model(b_input_ids,
                        token_type_ids=b_token_type_ids,
                        attention_mask=b_attention_mask,
                        labels = b_labels
                        )
        loss = outputs[0]
        preds = outputs[1]
        total_val_loss = total_val_loss + loss.item()

        val_preds = preds.detach().cpu().numpy()
        target_np = b_labels.detach().cpu().numpy()
        if j == 0:
            targets_list = target_np
            stacked_val_labels = val_preds
        else:
            targets_list = np.hstack((targets_list, target_np))
            stacked_val_labels = np.vstack((stacked_val_labels, val_preds))
    y_true = targets_list
    y_pred = np.argmax(stacked_val_labels, axis=1)
    val_acc = accuracy_score(y_true, y_pred)
    epoch_acc_scores_list.append(val_acc)

    print('Val loss:', total_val_loss)
    print('Val acc: ', val_acc)

    if epoch == 0:
        my_model_name = 'model_' + str(fold_index) + '.bin'
        torch.save(model.state_dict(), my_model_name)
        print('Saved model as ', my_model_name)
    else:
        val_acc_list = fold_val_acc_list[fold_index]
        best_val_acc = max(val_acc_list)

        if val_acc > best_val_acc:
            # save the model
            my_model_name = 'model_' + str(fold_index) + '.bin'
            torch.save(model.state_dict(), my_model_name)
            print('Val acc improved. Saved model as ', my_model_name)

    fold_val_acc_list[fold_index].append(val_acc)

cv_acc = sum(epoch_acc_scores_list) / 3
print("\nCV Acc:", cv_acc)
```