Machine Learning Assignment 2

By: Nikolaos Gournakis

Student ID: it22023

Loading the data

For the loading of the data I created the class MLProject2Dataset which is a subclass of the Dataset class, this is important so that we can easily use the Dataloader class later on.

```python
In [1]:  # Imports for MLProject2Dataset
         import glob
         import os
         import PIL.Image
         import pandas as pd
         import torchvision
         from torch.utils.data import Dataset
```

```python
In [2]:  class MLProject2Dataset(Dataset):
             # I added type-annotations to make the code more readable/usable
             def __init__(self, data_dir: str, metadata_fname='metadata.csv', transform: torchvision.transforms = None):
                 self.data_dir = data_dir
                 self.transform = transform

                 self.df = pd.DataFrame(columns=['image_id', 'path'])

                 # Get all files in data_dir
                 files = glob.glob(data_dir + '/*/*.jpg')
                 # If on windows, replace backslash with forward slash
                 # on linux, this does nothing
                 files = [file.replace(os.sep, '/') for file in files]
                 # The line commented bellow was used for testing on a smaller dataset
                 # files = files[:3000]
```

```python
        # Add all files to dataframe
        for file in files:
            # the `file` is in the format `data_dir/every/sub/directory/image_id.jpg`
            # that's why we split it by `/` and get the last element which will always be
            # `image_id.jpg`. Then just remove the `.jpg` extension
            image_id = file.split('/')[-1].split('.')[0]

            # `file` is the full path to the image, so we just create a local variable for readability
            path = file

            # Add new row to dataframe
            self.df.loc[len(self.df)] = [image_id, path]

        # Add metadata
        self.metadata = pd.read_csv(os.path.join(data_dir, metadata_fname))
        dx_categorical = pd.Categorical(self.metadata['dx'])

        # number_to_label_array is used to convert the prediction from a number back to
        # "Human Readable" format, this also used to make the confusion matrix have the labels as
        # cells instead of just numbers.
        # Which are the seven string labels given in the assignment
        self.number_to_label_array = [dx for dx in dx_categorical.categories]

        # dx_categorical.codes is a list of numbers from 0 to 6, which represent the seven labels
        # Effectively, we are converting the seven string labels to a number encoding from 0 to 6
        self.metadata['dx'] = dx_categorical.codes

        # Merge metadata with dataframe
        # This creates a new dataframe with all the columns from both dataframes
        # and merges them where the `image_id` is the same
        self.df = self.df.merge(self.metadata, on='image_id')

        # In the assignment, we are asked for the dataframe to have columns = ['image_id', 'path', 'dx']
        # so we drop the excess columns
        self.df = self.df.drop(columns=["lesion_id", "dx_type", "age", "sex", "localization"])

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

    def __getitem__(self, idx):
        # Get image path
```

```python
        image_path = self.df.iloc[idx]['path']

        # Load image
        image = PIL.Image.open(image_path)

        # Change pixel values to float
        # This is not really needed, but I do it just in case
        image = image.convert('RGB')

        # We can convert the image to tensor here or later during the data loading
        # image = torchvision.transforms.ToTensor()(image)

        # Get label
        label = self.df.iloc[idx]['dx']

        if self.transform:
            image = self.transform(image)

        return image, label
```

```python
In [3]:  # We use this function for the Simple CNN and the Complex CNN
         # with different values for M and N,but the rest of the code is the same
         def create_transform(M: int, N: int) -> torchvision.transforms:
             return torchvision.transforms.Compose([
                 torchvision.transforms.Resize((M, N)),
                 torchvision.transforms.ToTensor(),
                 torchvision.transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
             ])
```

train_net

```python
In [4]:  import torch
         from torch.utils.data import DataLoader
         from torch import nn, optim

         # These lists are used to store the loss and accuracy for each print_period
         # so that we can plot them later
         val_loss = []
         train_loss = []
```

```python
val_accuracy = []
train_accuracy = []

def train_net(model: nn.Module, trainloader: DataLoader, valloader: DataLoader = None, epochs: int = 10,
              optimizer: optim.Optimizer = None, loss: nn.modules.loss = None, device: str = 'cpu',
              print_period: int = 10) -> None:
    # Reset the lists so that we don't append to them if we train the model again / train another model
    global val_loss, train_loss, val_accuracy, train_accuracy
    val_loss = []
    train_loss = []
    val_accuracy = []
    train_accuracy = []

    for epoch in range(epochs):
        total = 0
        correct = 0
        running_loss = 0.0

        # batch is the index of the batch,
        # X is `BatchSize` number of images, y is `BatchSize` number of labels
        # In my experiments, I used `BatchSize` = 32
        for batch, (X, y) in enumerate(trainloader, 0):
            # Set model to train mode
            model.train()

            # Move the images and labels to the device specified
            X = X.to(device)
            y = y.to(device)

            # Zero the gradients
            optimizer.zero_grad()

            # Forward pass
            y_pred = model(X)

            # If I don't cast y to long, I get an error:
            # RuntimeError: "nll_loss_forward_reduce_cuda_kernel_2d_index" not implemented for 'Char'
            # solution: https://stackoverflow.com/a/71126544/13250408
            current_loss = loss(y_pred, y.long())

            # Backward pass
            current_loss.backward()
```

```python
            # Update weights
            optimizer.step()

            # Get the total number of images processed so far
            total += y.size(0)
            # Get the number of correct predictions so far
            correct += (y_pred.argmax(1) == y).type(torch.float).sum().item()
            # Get the running loss
            running_loss += current_loss.item()

            if batch % print_period == print_period - 1:

                avg_loss = running_loss / print_period
                avg_accuracy = correct / total * 100
                # Append the loss and accuracy for the current print_period
                train_loss.append(avg_loss)
                train_accuracy.append(avg_accuracy)
                # This is equal to `total` but for the validation set
                total_val = 0
                # This is equal to `correct` but for the validation set
                correct_val = 0
                # This is equal to `running_loss` but for the validation set
                running_loss_val = 0.0

                # If we have a validation set, we also want to print the validation loss and accuracy
                # noinspection GrazieInspection
                if valloader is not None:
                    # Set model to evaluation mode
                    model.eval()
                    # We don't want to update the weights, so we disable the gradient calculation
                    with torch.no_grad():
                        # Iterate over the validation set
                        for (X_val, y_val) in valloader:
                            # Move the images and labels to the device specified
                            X_val = X_val.to(device)
                            y_val = y_val.to(device)
                            # Get the predictions
                            y_pred_val = model(X_val)
                            # Get the total number of validation images processed so far
                            total_val += y_val.size(0)
                            # Get the running loss for the validation set
```

```
                    running_loss_val += loss(y_pred_val, y_val.long()).item()
                                # Get the number of correct predictions for the validation set
                    correct_val += (y_pred_val.argmax(1) == y_val).type(torch.float).sum().item()

                # To get the average loss, we divide the running loss by the `print_period`
                # to do the equivalent for the validation set, we need to do it differently
                # because we test all of the validation set at once, not in batches
                # so to get the average loss for the validation set, we divide the running loss
                # with the total number of batches in the validation set, which is equal to
                # the length of the validation loader
                avg_loss_val = running_loss_val / len(valloader)
                avg_accuracy_val = correct_val / total_val * 100
                # Append the loss and accuracy for the current print_period
                val_loss.append(avg_loss_val)
                val_accuracy.append(avg_accuracy_val)
                # Print the training loss, training accuracy, validation loss and validation accuracy
                print(
                    f'[{epoch + 1}, {batch + 1}] loss: {avg_loss} | accuracy: {avg_accuracy:.2f}% |'
                    f' val_loss: {avg_loss_val} | val_accuracy: {avg_accuracy_val:.2f}%')

            # If we don't have a validation set, we only print the training loss and accuracy
            else:
                print(f'[{epoch + 1}, {batch + 1}] loss: {avg_loss} | accuracy: {avg_accuracy :.4f}%')
            # Reset the running loss, total and correct for the next print_period
            running_loss = 0.0
            total = 0
            correct = 0
```

test_net

```python
In [32]:  from sklearn.metrics import confusion_matrix
          import seaborn as sns
          import matplotlib.pyplot as plt
          import numpy as np
          def test_net(model: nn.Module, testloader: DataLoader, number_to_label_array:list[str],
                       loss: nn.modules.loss = None, device: str = 'cpu',) -> None:
              # Set model to evaluation mode
              model.eval()
              correct = 0
              loss_test = 0.0
```

```python
        y_trues = []
        y_preds = []
        # We don't want to update the weights, so we disable the gradient calculation
        with torch.no_grad():
            # Iterate over the test set
            for (X, y) in testloader:
                # Move the images and labels to the device specified
                X = X.to(device)
                y = y.to(device)
                # Get the predictions
                y_pred_test = model(X)

                y_trues.extend(y.cpu().numpy())
                y_preds.extend(y_pred_test.argmax(1).cpu().numpy())
                # Get the total number of test images processed so far
                loss_test += loss(y_pred_test, y.long())
                # Get the number of correct predictions for the test set
                correct += (y_pred_test.argmax(1) == y).type(torch.float).sum().item()
        # Print the test loss and test accuracy
        print(f"Test loss: {loss_test / len(testloader)} |"
              f" Test accuracy: {correct / len(testloader.dataset) * 100:.2f}% ")

        # Create the confusion matrix
        conf_matrix = confusion_matrix(y_trues, y_preds)

        # Turn the confusion matrix into a dataframe for easier plotting
        df_cm = pd.DataFrame(conf_matrix, index=number_to_label_array,
                             columns=number_to_label_array)

        # Plot the confusion matrix
        sns.heatmap(df_cm, annot=True, fmt='g',cmap='PiYG')
        plt.xlabel('Predicted')
        plt.ylabel('True')
        plt.title('Confusion Matrix')
        plt.show()
```

```python
In [6]:  # Setting the device
         device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

Simple CNN

```python
In [7]: class SimpleConvNN(torch.nn.Module):
            def __init__(self):
                super(SimpleConvNN, self).__init__()
                # In the specification of the assignment, it says that
                # the images will be resized to [50 x 62] x 3 channels
                # the comments bellow are the output image size after each layer
                self.conv1 = torch.nn.Conv2d(3, 32, 3)  # 48 x 60 x 32
                # MaxPool 2x2 24 x 30 x 32
                self.conv2 = torch.nn.Conv2d(32, 64, 3)  # 22 x 28 x 64
                # MaxPool 2x2 11 x 14 x 64
                self.conv3 = torch.nn.Conv2d(64, 64, 3)  # 9 x 12 x 64
                # MaxPool 2x2 4 x 6 x 64
                self.fc = torch.nn.Linear(4 * 6 * 64, 7)
                self.pool = torch.nn.MaxPool2d(2)


            def forward(self, x):
                # Ordering the layers as given in the assignment
                x = self.pool(torch.nn.functional.relu(self.conv1(x)))
                x = self.pool(torch.nn.functional.relu(self.conv2(x)))
                x = self.pool(torch.nn.functional.relu(self.conv3(x)))
                x = torch.flatten(x, 1)
                x = self.fc(x)
                return x
```

Now let's train the model and test it

```python
In [8]: from torchsummary import summary

        # Constants from the assignment
        M = 50
        N = 62
        # Create the transforms
        dataset_transforms = create_transform(M, N)
        # Create the dataset
        dataset = MLProject2Dataset(data_dir='data', metadata_fname='metadata.csv', transform=dataset_transforms)
        # Split the dataset into train, validation and test
        train_dataset, val_dataset, test_dataset = torch.utils.data.random_split(dataset, [.6, .1, .3],
                                                                     generator=torch.Generator().manual_seed(42)
```

```python
                                                                                                      )
# Create the dataloaders
BATCH_SIZE = 32
trainloader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
valloader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=True)
testloader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=True)

# Create the model
model = SimpleConvNN()
model = model.to(device)
# Print the model summary
summary(model, (3, M, N))
# Create the optimizer
optimizer = optim.SGD(model.parameters(), lr=0.1)
# Create the loss function
loss = nn.CrossEntropyLoss()
# Train the model
train_net(model, trainloader, valloader, epochs=20, optimizer=optimizer, loss=loss, device=device)
# This cell took 38 minutes to run on my machine
```

```
----------------------------------------------------------------
        Layer (type)          Output Shape          Param #
================================================================
          Conv2d-1        [-1, 32, 48, 60]             896
       MaxPool2d-2        [-1, 32, 24, 30]               0
          Conv2d-3        [-1, 64, 22, 28]          18,496
       MaxPool2d-4        [-1, 64, 11, 14]               0
          Conv2d-5         [-1, 64, 9, 12]          36,928
       MaxPool2d-6          [-1, 64, 4, 6]               0
          Linear-7                 [-1, 7]          10,759
================================================================
Total params: 67,079
Trainable params: 67,079
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.04
Forward/backward pass size (MB): 1.32
Params size (MB): 0.26
Estimated Total Size (MB): 1.61
----------------------------------------------------------------
[1, 10] loss: 1.518793785572052 | accuracy: 58.13% | val_loss: 1.2464358545839787 | val_accuracy: 65.13%
[1, 20] loss: 1.3173851668834686 | accuracy: 68.75% | val_loss: 1.1391955614089966 | val_accuracy: 65.13%
[1, 30] loss: 1.1066274404525758 | accuracy: 66.25% | val_loss: 1.0768212135881186 | val_accuracy: 65.13%
[1, 40] loss: 1.0149292349815369 | accuracy: 69.38% | val_loss: 1.0525662787258625 | val_accuracy: 65.13%
[1, 50] loss: 1.139615136384964 | accuracy: 61.25% | val_loss: 1.1996386479586363 | val_accuracy: 65.33%
[1, 60] loss: 1.097195464372635 | accuracy: 65.62% | val_loss: 1.0418086061254144 | val_accuracy: 65.13%
[1, 70] loss: 1.0688795149326324 | accuracy: 65.94% | val_loss: 0.9865861423313618 | val_accuracy: 65.13%
[1, 80] loss: 0.9770754396915435 | accuracy: 67.81% | val_loss: 1.142379056662321 | val_accuracy: 65.13%
[1, 90] loss: 1.0007180273532867 | accuracy: 63.75% | val_loss: 1.0331340655684471 | val_accuracy: 65.13%
[1, 100] loss: 0.9722979307174683 | accuracy: 65.94% | val_loss: 1.2319824155420065 | val_accuracy: 65.33%
[1, 110] loss: 1.0412011325359345 | accuracy: 66.88% | val_loss: 0.9679132308810949 | val_accuracy: 65.43%
[1, 120] loss: 0.9695410609245301 | accuracy: 64.69% | val_loss: 0.958821153268218 | val_accuracy: 65.53%
[1, 130] loss: 0.9361404955387116 | accuracy: 66.25% | val_loss: 0.9314349181950092 | val_accuracy: 65.93%
[1, 140] loss: 1.058273833990097 | accuracy: 65.31% | val_loss: 0.950197359547019 | val_accuracy: 65.43%
[1, 150] loss: 0.8888957321643829 | accuracy: 68.12% | val_loss: 0.9497997667640448 | val_accuracy: 66.03%
[1, 160] loss: 0.9848702192306519 | accuracy: 66.88% | val_loss: 0.9146623685956001 | val_accuracy: 66.23%
[1, 170] loss: 0.9308148145675659 | accuracy: 68.12% | val_loss: 0.9495742432773113 | val_accuracy: 66.23%
[1, 180] loss: 0.8736093640327454 | accuracy: 70.94% | val_loss: 1.0582870300859213 | val_accuracy: 65.13%
[2, 10] loss: 0.8881802827119827 | accuracy: 70.62% | val_loss: 0.9386426508426666 | val_accuracy: 67.03%
[2, 20] loss: 0.9549904286861419 | accuracy: 66.25% | val_loss: 0.918483629822731 | val_accuracy: 65.73%
[2, 30] loss: 0.8058034777641296 | accuracy: 71.25% | val_loss: 0.9602652927860618 | val_accuracy: 65.93%
[2, 40] loss: 0.972988098859787 | accuracy: 65.94% | val_loss: 0.9043727740645409 | val_accuracy: 66.83%
```

```
[2,  50] loss: 0.8965781688690185 | accuracy: 68.12% | val_loss: 0.8962981477379799 | val_accuracy: 65.33%
[2,  60] loss: 0.9681354999542237 | accuracy: 66.25% | val_loss: 0.9129490694031119 | val_accuracy: 65.93%
[2,  70] loss: 0.8769855678081513 | accuracy: 70.00% | val_loss: 0.9951959066092968 | val_accuracy: 65.73%
[2,  80] loss: 0.8566119372844696 | accuracy: 70.31% | val_loss: 0.9078674167394638 | val_accuracy: 64.64%
[2,  90] loss: 0.8813894808292388 | accuracy: 70.31% | val_loss: 1.012676689773798 | val_accuracy: 66.03%
[2, 100] loss: 0.9491555988788605 | accuracy: 67.50% | val_loss: 0.8930072579532862 | val_accuracy: 66.33%
[2, 110] loss: 0.9263675212860107 | accuracy: 65.31% | val_loss: 0.891107683070004 | val_accuracy: 67.23%
[2, 120] loss: 0.9533082544803619 | accuracy: 68.12% | val_loss: 0.8923446051776409 | val_accuracy: 66.43%
[2, 130] loss: 0.9229607105255127 | accuracy: 64.38% | val_loss: 0.8439620733261108 | val_accuracy: 68.93%
[2, 140] loss: 0.8775909841060638 | accuracy: 68.12% | val_loss: 0.9100127127021551 | val_accuracy: 67.23%
[2, 150] loss: 1.0363579750061036 | accuracy: 63.44% | val_loss: 0.8825412821024656 | val_accuracy: 67.13%
[2, 160] loss: 0.7596043646335602 | accuracy: 73.12% | val_loss: 0.8908407762646675 | val_accuracy: 65.73%
[2, 170] loss: 0.8432924270629882 | accuracy: 70.00% | val_loss: 0.8675894299522042 | val_accuracy: 66.33%
[2, 180] loss: 0.973846846818924 | accuracy: 62.50% | val_loss: 0.8677255157381296 | val_accuracy: 66.93%
[3,  10] loss: 0.8693842113018035 | accuracy: 70.00% | val_loss: 0.9068197906017303 | val_accuracy: 68.33%
[3,  20] loss: 0.9245539367198944 | accuracy: 69.69% | val_loss: 0.8482250161468983 | val_accuracy: 66.43%
[3,  30] loss: 0.8058761715888977 | accuracy: 71.88% | val_loss: 0.8879662547260523 | val_accuracy: 66.33%
[3,  40] loss: 0.8018431305885315 | accuracy: 70.62% | val_loss: 0.8751243390142918 | val_accuracy: 68.73%
[3,  50] loss: 0.8726075828075409 | accuracy: 70.94% | val_loss: 1.0664616413414478 | val_accuracy: 64.44%
[3,  60] loss: 0.9015521109104156 | accuracy: 64.38% | val_loss: 0.8642947552725673 | val_accuracy: 68.33%
[3,  70] loss: 0.9306747198104859 | accuracy: 63.75% | val_loss: 0.8376922477036715 | val_accuracy: 68.03%
[3,  80] loss: 0.7847657382488251 | accuracy: 71.88% | val_loss: 0.8478030357509851 | val_accuracy: 67.03%
[3,  90] loss: 0.9061262130737304 | accuracy: 66.88% | val_loss: 0.8297563698142767 | val_accuracy: 68.93%
[3, 100] loss: 0.870602947473526 | accuracy: 66.56% | val_loss: 0.8714733850210905 | val_accuracy: 67.23%
[3, 110] loss: 0.8912302553653717 | accuracy: 66.25% | val_loss: 0.8362489119172096 | val_accuracy: 69.13%
[3, 120] loss: 0.8051473140716553 | accuracy: 71.88% | val_loss: 0.843952676281333 | val_accuracy: 68.83%
[3, 130] loss: 0.9509417712688446 | accuracy: 66.25% | val_loss: 0.8772696387022734 | val_accuracy: 69.43%
[3, 140] loss: 0.8863458216190339 | accuracy: 69.06% | val_loss: 0.8979610204696655 | val_accuracy: 65.53%
[3, 150] loss: 0.8993420749902725 | accuracy: 67.19% | val_loss: 0.9754824060946703 | val_accuracy: 68.53%
[3, 160] loss: 0.8832664847373962 | accuracy: 66.25% | val_loss: 0.8510226430371404 | val_accuracy: 68.03%
[3, 170] loss: 0.7881470412015915 | accuracy: 73.12% | val_loss: 0.9633007626980543 | val_accuracy: 61.24%
[3, 180] loss: 0.8733445048332215 | accuracy: 67.19% | val_loss: 0.9262318713590503 | val_accuracy: 65.63%
[4,  10] loss: 0.8964118480682373 | accuracy: 68.44% | val_loss: 0.8459508959203959 | val_accuracy: 66.93%
[4,  20] loss: 0.7726080775260925 | accuracy: 74.38% | val_loss: 0.8318781824782491 | val_accuracy: 67.53%
[4,  30] loss: 0.7636393129825592 | accuracy: 72.50% | val_loss: 0.8219802044332027 | val_accuracy: 68.33%
[4,  40] loss: 0.7814334988594055 | accuracy: 71.56% | val_loss: 0.8352053444832563 | val_accuracy: 67.13%
[4,  50] loss: 0.9669749915599823 | accuracy: 64.69% | val_loss: 0.9148869477212429 | val_accuracy: 67.03%
[4,  60] loss: 0.8185674667358398 | accuracy: 70.94% | val_loss: 0.8547702115029097 | val_accuracy: 68.83%
[4,  70] loss: 0.7825084030628204 | accuracy: 71.88% | val_loss: 0.8541211653500795 | val_accuracy: 68.73%
[4,  80] loss: 0.7922193884849549 | accuracy: 72.19% | val_loss: 0.8031186126172543 | val_accuracy: 68.43%
[4,  90] loss: 0.8617696523666382 | accuracy: 67.19% | val_loss: 0.8559656739234924 | val_accuracy: 67.73%
[4, 100] loss: 0.7899049997329712 | accuracy: 70.62% | val_loss: 0.8143490580841899 | val_accuracy: 69.23%
```

```
[4, 110] loss: 0.7909148812294007 | accuracy: 73.12% | val_loss: 0.839736714027822 | val_accuracy: 68.63%
[4, 120] loss: 0.9073679387569428 | accuracy: 69.38% | val_loss: 0.8112899251282215 | val_accuracy: 70.23%
[4, 130] loss: 0.86104910969734419 | accuracy: 67.81% | val_loss: 0.8687341436743736 | val_accuracy: 68.13%
[4, 140] loss: 0.8330842733383179 | accuracy: 69.38% | val_loss: 0.812447052448988 | val_accuracy: 70.33%
[4, 150] loss: 0.9180149853229522 | accuracy: 66.25% | val_loss: 0.8030542824417353 | val_accuracy: 70.13%
[4, 160] loss: 0.8498059511184692 | accuracy: 70.00% | val_loss: 0.8029752094298601 | val_accuracy: 70.63%
[4, 170] loss: 0.8327560901641846 | accuracy: 70.94% | val_loss: 0.7931884033605456 | val_accuracy: 70.53%
[4, 180] loss: 0.8189090669155121 | accuracy: 68.44% | val_loss: 0.7922591688111424 | val_accuracy: 69.93%
[5, 10] loss: 0.6943336248397827 | accuracy: 76.56% | val_loss: 0.8273032214492559 | val_accuracy: 70.33%
[5, 20] loss: 0.8296578228473663 | accuracy: 69.06% | val_loss: 0.8268737401813269 | val_accuracy: 67.03%
[5, 30] loss: 0.8139205902814866 | accuracy: 69.06% | val_loss: 0.8196424394845963 | val_accuracy: 69.53%
[5, 40] loss: 0.8774439513683319 | accuracy: 69.06% | val_loss: 0.8302071988582611 | val_accuracy: 68.33%
[5, 50] loss: 0.9478014469146728 | accuracy: 64.06% | val_loss: 0.8248966718092561 | val_accuracy: 70.23%
[5, 60] loss: 0.8615355134010315 | accuracy: 69.06% | val_loss: 0.8212379980832338 | val_accuracy: 69.23%
[5, 70] loss: 0.7926372230052948 | accuracy: 69.69% | val_loss: 1.0204764977097511 | val_accuracy: 62.14%
[5, 80] loss: 0.8106517910957336 | accuracy: 69.69% | val_loss: 0.7966296840459108 | val_accuracy: 69.83%
[5, 90] loss: 0.7340528309345246 | accuracy: 72.19% | val_loss: 0.7859837468713522 | val_accuracy: 71.03%
[5, 100] loss: 0.7939515560865402 | accuracy: 69.38% | val_loss: 0.8159124702215195 | val_accuracy: 69.13%
[5, 110] loss: 0.8378702044487 | accuracy: 69.06% | val_loss: 0.7837748015299439 | val_accuracy: 71.63%
[5, 120] loss: 0.8507627487182617 | accuracy: 71.56% | val_loss: 0.8011882528662682 | val_accuracy: 69.93%
[5, 130] loss: 0.7834202826023102 | accuracy: 71.25% | val_loss: 0.8648223374038935 | val_accuracy: 68.83%
[5, 140] loss: 0.7988188207149506 | accuracy: 72.50% | val_loss: 0.7861625403165817 | val_accuracy: 69.83%
[5, 150] loss: 0.7317732393741607 | accuracy: 73.75% | val_loss: 0.8254772759974003 | val_accuracy: 70.53%
[5, 160] loss: 0.7997078239917755 | accuracy: 70.00% | val_loss: 0.7750689536333084 | val_accuracy: 70.43%
[5, 170] loss: 0.9105548083782196 | accuracy: 64.38% | val_loss: 0.7684560809284449 | val_accuracy: 70.73%
[5, 180] loss: 0.8114823758602142 | accuracy: 74.06% | val_loss: 0.7881467174738646 | val_accuracy: 70.93%
[6, 10] loss: 0.7293206214904785 | accuracy: 73.75% | val_loss: 0.7530530923977494 | val_accuracy: 71.33%
[6, 20] loss: 0.7692529916763305 | accuracy: 69.06% | val_loss: 0.8168331105262041 | val_accuracy: 69.63%
[6, 30] loss: 0.6856015205383301 | accuracy: 74.69% | val_loss: 0.7832402596250176 | val_accuracy: 69.53%
[6, 40] loss: 0.7093329280614853 | accuracy: 74.38% | val_loss: 0.7900243597105145 | val_accuracy: 70.23%
[6, 50] loss: 0.7612624347209931 | accuracy: 72.19% | val_loss: 0.8026740187779069 | val_accuracy: 69.83%
[6, 60] loss: 0.7682025134563446 | accuracy: 70.31% | val_loss: 0.7949044657871127 | val_accuracy: 70.03%
[6, 70] loss: 0.8108818888664245 | accuracy: 70.31% | val_loss: 0.7516264021396637 | val_accuracy: 71.83%
[6, 80] loss: 0.8695433497428894 | accuracy: 68.44% | val_loss: 0.7606173325330019 | val_accuracy: 71.33%
[6, 90] loss: 0.8544639945030212 | accuracy: 68.12% | val_loss: 0.971032053232193 | val_accuracy: 60.54%
[6, 100] loss: 0.7887440085411072 | accuracy: 72.50% | val_loss: 0.7860160954296589 | val_accuracy: 70.53%
[6, 110] loss: 0.7232416808605194 | accuracy: 72.50% | val_loss: 0.7712192134931684 | val_accuracy: 72.13%
[6, 120] loss: 0.7800049692392349 | accuracy: 68.75% | val_loss: 0.7719339597970247 | val_accuracy: 70.73%
[6, 130] loss: 0.7867901444435119 | accuracy: 72.19% | val_loss: 0.7700664168223739 | val_accuracy: 70.03%
[6, 140] loss: 0.7703287363052368 | accuracy: 72.50% | val_loss: 0.8238052688539028 | val_accuracy: 69.83%
[6, 150] loss: 0.8538231074810028 | accuracy: 69.38% | val_loss: 0.7674029665067792 | val_accuracy: 71.03%
[6, 160] loss: 0.6847216010093689 | accuracy: 73.44% | val_loss: 0.8001913335174322 | val_accuracy: 71.33%
```

```
[6, 170] loss: 0.7412172734737397 | accuracy: 72.81% | val_loss: 0.7916136980056763 | val_accuracy: 69.93%
[6, 180] loss: 0.8604855358600616 | accuracy: 69.38% | val_loss: 0.7771682403981686 | val_accuracy: 72.53%
[7, 10] loss: 0.7201659739017486 | accuracy: 72.19% | val_loss: 0.7655354756861925 | val_accuracy: 71.53%
[7, 20] loss: 0.8270796656608581 | accuracy: 70.62% | val_loss: 0.7727065803483129 | val_accuracy: 71.13%
[7, 30] loss: 0.7508438289165497 | accuracy: 70.94% | val_loss: 0.7667716834694147 | val_accuracy: 71.33%
[7, 40] loss: 0.7167212486267089 | accuracy: 73.75% | val_loss: 0.7659776322543621 | val_accuracy: 71.53%
[7, 50] loss: 0.7115629225969314 | accuracy: 76.25% | val_loss: 0.8114396939054132 | val_accuracy: 71.73%
[7, 60] loss: 0.8154988765716553 | accuracy: 69.38% | val_loss: 0.7904073316603899 | val_accuracy: 71.43%
[7, 70] loss: 0.6974685281515122 | accuracy: 75.62% | val_loss: 0.7728052232414484 | val_accuracy: 70.93%
[7, 80] loss: 0.8187405109405518 | accuracy: 69.38% | val_loss: 0.759864516556263 | val_accuracy: 70.73%
[7, 90] loss: 0.7157119929790496 | accuracy: 75.31% | val_loss: 0.7406004630029202 | val_accuracy: 72.53%
[7, 100] loss: 0.5760733723640442 | accuracy: 77.19% | val_loss: 0.9259790424257517 | val_accuracy: 66.13%
[7, 110] loss: 0.7420385748147964 | accuracy: 73.44% | val_loss: 0.7510803192853928 | val_accuracy: 70.23%
[7, 120] loss: 0.7888030230998992 | accuracy: 70.62% | val_loss: 0.7914383923634887 | val_accuracy: 69.63%
[7, 130] loss: 0.8440999865531922 | accuracy: 66.88% | val_loss: 0.7471137382090092 | val_accuracy: 70.83%
[7, 140] loss: 0.7159431368112564 | accuracy: 72.50% | val_loss: 0.7437009466812015 | val_accuracy: 71.43%
[7, 150] loss: 0.8987858295440674 | accuracy: 68.75% | val_loss: 0.7517248010262847 | val_accuracy: 71.33%
[7, 160] loss: 0.7482125192880631 | accuracy: 70.31% | val_loss: 0.7395641524344683 | val_accuracy: 72.03%
[7, 170] loss: 0.7596843123435975 | accuracy: 70.00% | val_loss: 0.7621422903612256 | val_accuracy: 71.33%
[7, 180] loss: 0.7990598022937775 | accuracy: 72.19% | val_loss: 0.7248687986284494 | val_accuracy: 71.83%
[8, 10] loss: 0.7472058653831481 | accuracy: 74.38% | val_loss: 0.7357574244961143 | val_accuracy: 71.63%
[8, 20] loss: 0.7074092537164688 | accuracy: 71.88% | val_loss: 0.7107342332601547 | val_accuracy: 73.03%
[8, 30] loss: 0.7300812602043152 | accuracy: 72.50% | val_loss: 0.7435651160776615 | val_accuracy: 72.53%
[8, 40] loss: 0.6924116909503937 | accuracy: 74.69% | val_loss: 0.7405048850923777 | val_accuracy: 72.53%
[8, 50] loss: 0.7438171476125717 | accuracy: 72.19% | val_loss: 0.734918893314898 | val_accuracy: 72.63%
[8, 60] loss: 0.7346929490566254 | accuracy: 70.31% | val_loss: 0.8128311792388558 | val_accuracy: 67.73%
[8, 70] loss: 0.790061891078949 | accuracy: 67.50% | val_loss: 0.7798128118738532 | val_accuracy: 70.33%
[8, 80] loss: 0.8174928545951843 | accuracy: 65.31% | val_loss: 0.7315768972039223 | val_accuracy: 72.33%
[8, 90] loss: 0.7609184622764588 | accuracy: 72.50% | val_loss: 0.7726689288392663 | val_accuracy: 70.93%
[8, 100] loss: 0.7715620577335358 | accuracy: 71.56% | val_loss: 0.7555542076006532 | val_accuracy: 72.13%
[8, 110] loss: 0.7240578949451446 | accuracy: 73.75% | val_loss: 0.7182535585016012 | val_accuracy: 72.13%
[8, 120] loss: 0.7684918165206909 | accuracy: 71.88% | val_loss: 0.7245315983891487 | val_accuracy: 71.83%
[8, 130] loss: 0.6249454170465469 | accuracy: 80.00% | val_loss: 0.7145494939759374 | val_accuracy: 72.43%
[8, 140] loss: 0.6757012993097306 | accuracy: 74.06% | val_loss: 0.7314581628888845 | val_accuracy: 72.93%
[8, 150] loss: 0.7642005205154419 | accuracy: 73.44% | val_loss: 0.7061256598681211 | val_accuracy: 73.13%
[8, 160] loss: 0.7017618745565415 | accuracy: 75.31% | val_loss: 0.7944187261164188 | val_accuracy: 71.83%
[8, 170] loss: 0.7317767053842544 | accuracy: 77.19% | val_loss: 0.7853284943848848 | val_accuracy: 71.43%
[8, 180] loss: 0.7986599743366242 | accuracy: 68.44% | val_loss: 0.7153360703960061 | val_accuracy: 72.53%
[9, 10] loss: 0.7119902431964874 | accuracy: 75.31% | val_loss: 0.7258126339875162 | val_accuracy: 72.23%
[9, 20] loss: 0.7213380724191666 | accuracy: 72.81% | val_loss: 0.7634491845965385 | val_accuracy: 73.43%
[9, 30] loss: 0.7433912038803101 | accuracy: 73.75% | val_loss: 0.7658159993588924 | val_accuracy: 71.33%
[9, 40] loss: 0.7678873538970947 | accuracy: 71.88% | val_loss: 0.7220282908529043 | val_accuracy: 71.63%
```

```
[9, 50] loss: 0.6558604896068573 | accuracy: 75.31% | val_loss: 0.7056128429248929 | val_accuracy: 72.93%
[9, 60] loss: 0.7123411208391189 | accuracy: 74.38% | val_loss: 0.7661803336814046 | val_accuracy: 71.73%
[9, 70] loss: 0.7399612456560135 | accuracy: 72.19% | val_loss: 0.7125594140961766 | val_accuracy: 72.33%
[9, 80] loss: 0.8456421494483948 | accuracy: 67.50% | val_loss: 0.7448426224291325 | val_accuracy: 73.13%
[9, 90] loss: 0.6530835926532745 | accuracy: 77.19% | val_loss: 0.7160550113767385 | val_accuracy: 71.53%
[9, 100] loss: 0.710214251279831 | accuracy: 75.94% | val_loss: 0.7123290412127972 | val_accuracy: 72.93%
[9, 110] loss: 0.6917033508419991 | accuracy: 73.12% | val_loss: 0.8749028444290161 | val_accuracy: 65.63%
[9, 120] loss: 0.7414666444063187 | accuracy: 71.25% | val_loss: 0.73248325381428 | val_accuracy: 72.13%
[9, 130] loss: 0.7209202706813812 | accuracy: 72.19% | val_loss: 0.729455335997045 | val_accuracy: 72.33%
[9, 140] loss: 0.755241310596466 | accuracy: 71.56% | val_loss: 0.6835456592962146 | val_accuracy: 73.73%
[9, 150] loss: 0.7318120658397674 | accuracy: 73.75% | val_loss: 0.7952764229848981 | val_accuracy: 70.63%
[9, 160] loss: 0.657338747382164 | accuracy: 77.50% | val_loss: 0.7002133000642061 | val_accuracy: 73.43%
[9, 170] loss: 0.6631748795509338 | accuracy: 75.94% | val_loss: 0.7222733767703176 | val_accuracy: 73.63%
[9, 180] loss: 0.6594358712434769 | accuracy: 73.12% | val_loss: 0.7528277151286602 | val_accuracy: 72.13%
[10, 10] loss: 0.7912569761276245 | accuracy: 68.75% | val_loss: 0.765833580866456 | val_accuracy: 72.73%
[10, 20] loss: 0.67460917532444 | accuracy: 75.62% | val_loss: 0.7383092353120446 | val_accuracy: 72.03%
[10, 30] loss: 0.7668874740600586 | accuracy: 71.88% | val_loss: 0.7020319998264313 | val_accuracy: 73.53%
[10, 40] loss: 0.5933469235897064 | accuracy: 76.56% | val_loss: 0.6806788612157106 | val_accuracy: 73.33%
[10, 50] loss: 0.6881805062294006 | accuracy: 78.75% | val_loss: 0.7576083978638053 | val_accuracy: 73.03%
[10, 60] loss: 0.6805338025093078 | accuracy: 74.38% | val_loss: 0.706424112431705 | val_accuracy: 72.03%
[10, 70] loss: 0.7163186699151993 | accuracy: 76.25% | val_loss: 0.7252379199489951 | val_accuracy: 72.53%
[10, 80] loss: 0.6529519617557525 | accuracy: 74.06% | val_loss: 0.7161224232986569 | val_accuracy: 72.73%
[10, 90] loss: 0.7548629224300385 | accuracy: 71.88% | val_loss: 0.6887117270380259 | val_accuracy: 73.03%
[10, 100] loss: 0.6154628098011017 | accuracy: 77.50% | val_loss: 0.7806516317650676 | val_accuracy: 70.63%
[10, 110] loss: 0.7019013822078705 | accuracy: 77.19% | val_loss: 0.6912295473739505 | val_accuracy: 74.73%
[10, 120] loss: 0.7110175430774689 | accuracy: 75.31% | val_loss: 0.7406770568341017 | val_accuracy: 72.93%
[10, 130] loss: 0.7096666127443314 | accuracy: 74.38% | val_loss: 0.6961056375876069 | val_accuracy: 72.73%
[10, 140] loss: 0.7037199079990387 | accuracy: 75.31% | val_loss: 0.7036279514431953 | val_accuracy: 71.93%
[10, 150] loss: 0.7470318853855134 | accuracy: 71.56% | val_loss: 0.7296681785956025 | val_accuracy: 72.93%
[10, 160] loss: 0.6724254995584488 | accuracy: 75.00% | val_loss: 0.7148599708452821 | val_accuracy: 73.33%
[10, 170] loss: 0.6460056304931641 | accuracy: 78.12% | val_loss: 0.7627247888594866 | val_accuracy: 73.13%
[10, 180] loss: 0.6986802279949188 | accuracy: 71.88% | val_loss: 0.6879905685782433 | val_accuracy: 74.73%
[11, 10] loss: 0.6793292552232743 | accuracy: 72.19% | val_loss: 0.6922942642122507 | val_accuracy: 74.43%
[11, 20] loss: 0.7512198835611343 | accuracy: 71.25% | val_loss: 0.715288084000349 | val_accuracy: 73.63%
[11, 30] loss: 0.6893922328948975 | accuracy: 75.94% | val_loss: 0.7039272990077734 | val_accuracy: 74.33%
[11, 40] loss: 0.6532567828893662 | accuracy: 76.25% | val_loss: 0.7341353902593255 | val_accuracy: 71.53%
[11, 50] loss: 0.5727572709321975 | accuracy: 80.00% | val_loss: 0.7120752474293113 | val_accuracy: 73.73%
[11, 60] loss: 0.7334563255310058 | accuracy: 73.12% | val_loss: 0.7163863126188517 | val_accuracy: 73.33%
[11, 70] loss: 0.7279835641384125 | accuracy: 76.25% | val_loss: 0.6962894788011909 | val_accuracy: 73.43%
[11, 80] loss: 0.7550597846508026 | accuracy: 70.00% | val_loss: 0.6997461346909404 | val_accuracy: 73.13%
[11, 90] loss: 0.708294153213501 | accuracy: 74.69% | val_loss: 0.7009940827265382 | val_accuracy: 74.03%
[11, 100] loss: 0.6725185066461563 | accuracy: 76.56% | val_loss: 0.7723301602527499 | val_accuracy: 70.63%
```

```
[11, 110] loss: 0.6225936442613602 | accuracy: 76.56% | val_loss: 0.7482059467583895 | val_accuracy: 73.93%
[11, 120] loss: 0.6968424797058106 | accuracy: 71.56% | val_loss: 0.6819869503378868 | val_accuracy: 71.93%
[11, 130] loss: 0.7123645216226577 | accuracy: 74.69% | val_loss: 0.7200260497629642 | val_accuracy: 74.03%
[11, 140] loss: 0.6579011231660843 | accuracy: 76.56% | val_loss: 0.7896305872127414 | val_accuracy: 74.33%
[11, 150] loss: 0.5555790215730667 | accuracy: 77.81% | val_loss: 0.7146493038162589 | val_accuracy: 73.83%
[11, 160] loss: 0.7250492215156555 | accuracy: 76.56% | val_loss: 0.7603372680023313 | val_accuracy: 72.23%
[11, 170] loss: 0.736522427201271 | accuracy: 74.38% | val_loss: 0.8144238069653511 | val_accuracy: 70.33%
[11, 180] loss: 0.6190130889415741 | accuracy: 78.75% | val_loss: 0.6995085570961237 | val_accuracy: 74.63%
[12, 10] loss: 0.7211729735136032 | accuracy: 74.06% | val_loss: 0.7117769010365009 | val_accuracy: 74.33%
[12, 20] loss: 0.6274710267782211 | accuracy: 75.62% | val_loss: 0.7002258412539959 | val_accuracy: 73.93%
[12, 30] loss: 0.5933802843093872 | accuracy: 79.06% | val_loss: 0.7749050380662084 | val_accuracy: 72.63%
[12, 40] loss: 0.6219657510519028 | accuracy: 76.88% | val_loss: 0.6764179794117808 | val_accuracy: 74.53%
[12, 50] loss: 0.6278844922780991 | accuracy: 77.50% | val_loss: 0.6724238786846399 | val_accuracy: 74.23%
[12, 60] loss: 0.6802094221115113 | accuracy: 74.06% | val_loss: 0.6892040204256773 | val_accuracy: 73.93%
[12, 70] loss: 0.7436098158359528 | accuracy: 72.50% | val_loss: 0.8002385254949331 | val_accuracy: 70.43%
[12, 80] loss: 0.6353487700223923 | accuracy: 78.12% | val_loss: 0.709632471203804 | val_accuracy: 73.23%
[12, 90] loss: 0.5829992562532424 | accuracy: 78.44% | val_loss: 0.685943434946239 | val_accuracy: 74.33%
[12, 100] loss: 0.5867188215255738 | accuracy: 76.88% | val_loss: 0.710184920579195 | val_accuracy: 72.93%
[12, 110] loss: 0.6275329828262329 | accuracy: 77.50% | val_loss: 0.8252037083730102 | val_accuracy: 72.33%
[12, 120] loss: 0.6847056448459625 | accuracy: 75.94% | val_loss: 0.667448160238564 | val_accuracy: 74.43%
[12, 130] loss: 0.6179657518863678 | accuracy: 76.56% | val_loss: 0.7003251137211919 | val_accuracy: 73.83%
[12, 140] loss: 0.6324828088283538 | accuracy: 74.69% | val_loss: 0.7060683388262987 | val_accuracy: 74.33%
[12, 150] loss: 0.681729644536972 | accuracy: 77.50% | val_loss: 0.6884771967306733 | val_accuracy: 74.03%
[12, 160] loss: 0.7363033652305603 | accuracy: 73.75% | val_loss: 0.6894932454451919 | val_accuracy: 73.93%
[12, 170] loss: 0.6367806375026703 | accuracy: 73.75% | val_loss: 0.7542187459766865 | val_accuracy: 72.33%
[12, 180] loss: 0.6397121131420136 | accuracy: 76.25% | val_loss: 0.7097441339865327 | val_accuracy: 73.83%
[13, 10] loss: 0.6552366822957992 | accuracy: 77.19% | val_loss: 0.7108350098133087 | val_accuracy: 73.13%
[13, 20] loss: 0.6448940813541413 | accuracy: 72.81% | val_loss: 0.659646418876946 | val_accuracy: 74.73%
[13, 30] loss: 0.6685197502374649 | accuracy: 72.81% | val_loss: 0.6622613053768873 | val_accuracy: 72.93%
[13, 40] loss: 0.5990982353687286 | accuracy: 76.56% | val_loss: 0.6766712348908186 | val_accuracy: 74.43%
[13, 50] loss: 0.619576495885849 | accuracy: 73.75% | val_loss: 0.6668750289827585 | val_accuracy: 74.63%
[13, 60] loss: 0.6077278643846512 | accuracy: 76.25% | val_loss: 0.7763166483491659 | val_accuracy: 72.33%
[13, 70] loss: 0.662548103928566 | accuracy: 75.31% | val_loss: 0.6663905223831534 | val_accuracy: 74.73%
[13, 80] loss: 0.5698891848325729 | accuracy: 78.44% | val_loss: 0.7176613211631775 | val_accuracy: 73.03%
[13, 90] loss: 0.6781819313764572 | accuracy: 74.06% | val_loss: 0.6956408619880676 | val_accuracy: 73.43%
[13, 100] loss: 0.7258035182952881 | accuracy: 75.31% | val_loss: 0.7739758202806115 | val_accuracy: 72.73%
[13, 110] loss: 0.7156279653310775 | accuracy: 73.44% | val_loss: 0.7422378566116095 | val_accuracy: 71.63%
[13, 120] loss: 0.6231303006410599 | accuracy: 75.31% | val_loss: 0.6996891964226961 | val_accuracy: 74.43%
[13, 130] loss: 0.6647371232509613 | accuracy: 77.50% | val_loss: 0.7035988587886095 | val_accuracy: 74.93%
[13, 140] loss: 0.6621907830238343 | accuracy: 75.94% | val_loss: 0.7811008766293526 | val_accuracy: 71.83%
[13, 150] loss: 0.6543887615203857 | accuracy: 75.94% | val_loss: 0.7298274524509907 | val_accuracy: 73.43%
[13, 160] loss: 0.6028830647468567 | accuracy: 80.00% | val_loss: 0.7104420308023691 | val_accuracy: 74.03%
```

```
[13, 170] loss: 0.5485974431037903 | accuracy: 78.44% | val_loss: 0.7170111071318388 | val_accuracy: 73.93%
[13, 180] loss: 0.656700250506401 | accuracy: 78.75% | val_loss: 0.7163003506138921 | val_accuracy: 72.93%
[14, 10] loss: 0.6210664510726929 | accuracy: 76.88% | val_loss: 0.6913860877975821 | val_accuracy: 74.63%
[14, 20] loss: 0.5968304127454758 | accuracy: 76.88% | val_loss: 0.8990829922258854 | val_accuracy: 64.44%
[14, 30] loss: 0.5854073107242584 | accuracy: 78.75% | val_loss: 0.7114640651270747 | val_accuracy: 74.93%
[14, 40] loss: 0.6887169867753983 | accuracy: 73.44% | val_loss: 0.6752008963376284 | val_accuracy: 75.42%
[14, 50] loss: 0.6075867474079132 | accuracy: 77.50% | val_loss: 0.7140157101675868 | val_accuracy: 73.63%
[14, 60] loss: 0.5836057841777802 | accuracy: 80.00% | val_loss: 0.7193344058468938 | val_accuracy: 74.73%
[14, 70] loss: 0.639396271109581 | accuracy: 75.31% | val_loss: 0.7545861825346947 | val_accuracy: 70.33%
[14, 80] loss: 0.6121407896280289 | accuracy: 76.88% | val_loss: 0.65510512329638 | val_accuracy: 75.92%
[14, 90] loss: 0.640603193640709 | accuracy: 75.31% | val_loss: 0.6859643682837486 | val_accuracy: 74.83%
[14, 100] loss: 0.5649170696735382 | accuracy: 77.50% | val_loss: 0.6660568313673139 | val_accuracy: 75.02%
[14, 110] loss: 0.6906864762306213 | accuracy: 72.81% | val_loss: 0.6848413050174713 | val_accuracy: 74.13%
[14, 120] loss: 0.6257061004638672 | accuracy: 79.38% | val_loss: 0.7237070016562939 | val_accuracy: 72.83%
[14, 130] loss: 0.6568994432687759 | accuracy: 76.25% | val_loss: 0.8098530760034919 | val_accuracy: 66.73%
[14, 140] loss: 0.6387811183929444 | accuracy: 75.94% | val_loss: 0.7568588331341743 | val_accuracy: 72.83%
[14, 150] loss: 0.613511860370636 | accuracy: 79.69% | val_loss: 0.7161957258358598 | val_accuracy: 73.93%
[14, 160] loss: 0.580518102645874 | accuracy: 77.50% | val_loss: 0.672249655239284 | val_accuracy: 75.22%
[14, 170] loss: 0.5919616669416428 | accuracy: 80.94% | val_loss: 0.844207962974906 | val_accuracy: 68.83%
[14, 180] loss: 0.620694363117218 | accuracy: 75.31% | val_loss: 0.6816788255237043 | val_accuracy: 73.63%
[15, 10] loss: 0.7027302950620651 | accuracy: 71.88% | val_loss: 0.6635789200663567 | val_accuracy: 74.43%
[15, 20] loss: 0.6318568706512451 | accuracy: 76.25% | val_loss: 0.6924440702423453 | val_accuracy: 73.93%
[15, 30] loss: 0.5564908057451248 | accuracy: 78.12% | val_loss: 0.6932139787822962 | val_accuracy: 74.33%
[15, 40] loss: 0.5283477246761322 | accuracy: 80.00% | val_loss: 0.7637495808303356 | val_accuracy: 69.43%
[15, 50] loss: 0.49295041859149935 | accuracy: 80.62% | val_loss: 0.7047851569950581 | val_accuracy: 76.12%
[15, 60] loss: 0.48612076342105864 | accuracy: 83.44% | val_loss: 0.7555817607790232 | val_accuracy: 74.33%
[15, 70] loss: 0.6971864491701126 | accuracy: 75.94% | val_loss: 0.6864071409218013 | val_accuracy: 74.43%
[15, 80] loss: 0.649814561009407 | accuracy: 75.62% | val_loss: 0.7215990936383605 | val_accuracy: 73.63%
[15, 90] loss: 0.6081157863140106 | accuracy: 77.81% | val_loss: 0.6754725826904178 | val_accuracy: 74.93%
[15, 100] loss: 0.5660370171070099 | accuracy: 82.19% | val_loss: 0.7236782675608993 | val_accuracy: 72.93%
[15, 110] loss: 0.5700437128543854 | accuracy: 79.06% | val_loss: 0.6828859313391149 | val_accuracy: 75.02%
[15, 120] loss: 0.7321209251880646 | accuracy: 70.94% | val_loss: 0.7124384213238955 | val_accuracy: 74.73%
[15, 130] loss: 0.604701042175293 | accuracy: 76.88% | val_loss: 0.667386676184833 | val_accuracy: 75.62%
[15, 140] loss: 0.5418462812900543 | accuracy: 78.44% | val_loss: 0.7073777318000793 | val_accuracy: 74.73%
[15, 150] loss: 0.5294135987758637 | accuracy: 80.00% | val_loss: 0.7401409512385726 | val_accuracy: 74.33%
[15, 160] loss: 0.626316973567009 | accuracy: 78.44% | val_loss: 0.6848354106768966 | val_accuracy: 75.52%
[15, 170] loss: 0.6773416697978973 | accuracy: 73.44% | val_loss: 0.6725482847541571 | val_accuracy: 74.03%
[15, 180] loss: 0.56961729824543 | accuracy: 76.25% | val_loss: 0.6913834568113089 | val_accuracy: 74.53%
[16, 10] loss: 0.5964781373739243 | accuracy: 79.69% | val_loss: 0.6999955726787448 | val_accuracy: 73.23%
[16, 20] loss: 0.6769651979207992 | accuracy: 75.31% | val_loss: 0.7048727152869105 | val_accuracy: 73.43%
[16, 30] loss: 0.6941670686006546 | accuracy: 74.38% | val_loss: 0.6801823717541993 | val_accuracy: 74.13%
[16, 40] loss: 0.5968503683805466 | accuracy: 78.44% | val_loss: 0.7077294941991568 | val_accuracy: 73.83%
```

```
[16, 50] loss: 0.5328584462404251 | accuracy: 78.12% | val_loss: 0.701907797716558 | val_accuracy: 74.53%
[16, 60] loss: 0.6565305113792419 | accuracy: 73.44% | val_loss: 0.6783710746094584 | val_accuracy: 75.02%
[16, 70] loss: 0.6286000639200211 | accuracy: 79.06% | val_loss: 0.7694631535559893 | val_accuracy: 69.43%
[16, 80] loss: 0.5319704949855805 | accuracy: 81.25% | val_loss: 0.6691893581300974 | val_accuracy: 75.62%
[16, 90] loss: 0.5211558490991592 | accuracy: 83.12% | val_loss: 0.9185375943779945 | val_accuracy: 72.73%
[16, 100] loss: 0.6595533668994904 | accuracy: 75.00% | val_loss: 0.697311645373702 | val_accuracy: 73.63%
[16, 110] loss: 0.6359914600849151 | accuracy: 77.81% | val_loss: 0.682224134914577 | val_accuracy: 73.93%
[16, 120] loss: 0.460530486702919 | accuracy: 84.06% | val_loss: 0.7135714227333665 | val_accuracy: 73.93%
[16, 130] loss: 0.5834781587123871 | accuracy: 77.19% | val_loss: 0.6908708298578858 | val_accuracy: 73.93%
[16, 140] loss: 0.49966336488723756 | accuracy: 82.19% | val_loss: 0.7195969270542264 | val_accuracy: 74.53%
[16, 150] loss: 0.5075605839490891 | accuracy: 79.69% | val_loss: 0.66161846090 10816 | val_accuracy: 75.52%
[16, 160] loss: 0.5394823312759399 | accuracy: 77.81% | val_loss: 0.6931201461702585 | val_accuracy: 75.42%
[16, 170] loss: 0.5480892419815063 | accuracy: 81.25% | val_loss: 0.678401050157845 | val_accuracy: 75.72%
[16, 180] loss: 0.539176344871521 | accuracy: 80.62% | val_loss: 0.6582878576591611 | val_accuracy: 75.82%
[17, 10] loss: 0.5663069456815719 | accuracy: 79.06% | val_loss: 0.6348041575402021 | val_accuracy: 76.22%
[17, 20] loss: 0.5843781590461731 | accuracy: 78.75% | val_loss: 0.9439464490860701 | val_accuracy: 62.44%
[17, 30] loss: 0.5426694095134735 | accuracy: 80.00% | val_loss: 0.654281809926033 | val_accuracy: 75.92%
[17, 40] loss: 0.4895738273859024 | accuracy: 84.06% | val_loss: 0.6694999532774091 | val_accuracy: 75.52%
[17, 50] loss: 0.6127139866352082 | accuracy: 76.88% | val_loss: 0.6823668917641044 | val_accuracy: 75.52%
[17, 60] loss: 0.5314958095550537 | accuracy: 79.38% | val_loss: 0.6546584209427238 | val_accuracy: 75.92%
[17, 70] loss: 0.4693871021270752 | accuracy: 83.12% | val_loss: 0.778367917984724 | val_accuracy: 74.63%
[17, 80] loss: 0.4597057715058327 | accuracy: 83.44% | val_loss: 0.6755098002031446 | val_accuracy: 76.02%
[17, 90] loss: 0.534672400355339 | accuracy: 81.56% | val_loss: 0.6665982119739056 | val_accuracy: 75.02%
[17, 100] loss: 0.5699723839759827 | accuracy: 77.81% | val_loss: 0.6565159317106009 | val_accuracy: 76.12%
[17, 110] loss: 0.6394007474184036 | accuracy: 77.81% | val_loss: 0.6784810954704881 | val_accuracy: 74.73%
[17, 120] loss: 0.6771276950836181 | accuracy: 75.31% | val_loss: 0.6635330151766539 | val_accuracy: 75.82%
[17, 130] loss: 0.5809164106845855 | accuracy: 80.62% | val_loss: 0.6598292263224721 | val_accuracy: 75.42%
[17, 140] loss: 0.5501100093126297 | accuracy: 77.50% | val_loss: 0.7266619727015495 | val_accuracy: 74.53%
[17, 150] loss: 0.5182750701904297 | accuracy: 79.06% | val_loss: 0.6688973223790526 | val_accuracy: 75.52%
[17, 160] loss: 0.5048647135496139 | accuracy: 81.56% | val_loss: 0.7044289372861385 | val_accuracy: 72.93%
[17, 170] loss: 0.6183887988328933 | accuracy: 75.94% | val_loss: 0.7290096711367369 | val_accuracy: 74.83%
[17, 180] loss: 0.5638150960206986 | accuracy: 80.00% | val_loss: 0.7700883261859417 | val_accuracy: 70.13%
[18, 10] loss: 0.4851915866136551 | accuracy: 83.75% | val_loss: 0.6997059807181358 | val_accuracy: 75.72%
[18, 20] loss: 0.559866976737976 | accuracy: 78.44% | val_loss: 0.6820231368765235 | val_accuracy: 74.23%
[18, 30] loss: 0.530940717458725 | accuracy: 82.19% | val_loss: 0.6902415594086051 | val_accuracy: 75.22%
[18, 40] loss: 0.5704767525196075 | accuracy: 82.19% | val_loss: 0.6783897113054991 | val_accuracy: 75.72%
[18, 50] loss: 0.5370629787445068 | accuracy: 81.88% | val_loss: 0.6840983824804425 | val_accuracy: 75.12%
[18, 60] loss: 0.4762289136648178 | accuracy: 81.88% | val_loss: 0.6804696870967746 | val_accuracy: 74.33%
[18, 70] loss: 0.506780418753624 | accuracy: 80.00% | val_loss: 0.6630978919565678 | val_accuracy: 75.62%
[18, 80] loss: 0.5853408306837082 | accuracy: 78.44% | val_loss: 0.685639831237495 | val_accuracy: 74.43%
[18, 90] loss: 0.569967982172966 | accuracy: 79.38% | val_loss: 0.7412049937993288 | val_accuracy: 73.03%
[18, 100] loss: 0.423202720284462 | accuracy: 84.06% | val_loss: 0.7597796665504575 | val_accuracy: 74.83%
```

```
[18, 110] loss: 0.557905125617981 | accuracy: 79.38% | val_loss: 0.7167279990389943 | val_accuracy: 74.73%
[18, 120] loss: 0.4835855334997177 | accuracy: 84.06% | val_loss: 0.6999374832957983 | val_accuracy: 73.33%
[18, 130] loss: 0.5112580478191375 | accuracy: 81.88% | val_loss: 0.7543759401887655 | val_accuracy: 72.33%
[18, 140] loss: 0.5205389887094498 | accuracy: 80.31% | val_loss: 0.7049632994458079 | val_accuracy: 75.52%
[18, 150] loss: 0.5457695126533508 | accuracy: 77.81% | val_loss: 0.6737318746745586 | val_accuracy: 74.93%
[18, 160] loss: 0.5713148415088654 | accuracy: 79.06% | val_loss: 0.6971875838935375 | val_accuracy: 75.32%
[18, 170] loss: 0.5451025307178498 | accuracy: 80.00% | val_loss: 0.765893530100584 | val_accuracy: 70.43%
[18, 180] loss: 0.5250411629676819 | accuracy: 80.00% | val_loss: 0.6648302497342229 | val_accuracy: 74.63%
[19, 10] loss: 0.5170228570699692 | accuracy: 81.25% | val_loss: 0.703082007355988 | val_accuracy: 75.32%
[19, 20] loss: 0.4787963777780533 | accuracy: 82.81% | val_loss: 0.7045997567474842 | val_accuracy: 74.03%
[19, 30] loss: 0.5419222474098205 | accuracy: 81.25% | val_loss: 0.6788358530029655 | val_accuracy: 76.22%
[19, 40] loss: 0.4324310928583145 | accuracy: 84.38% | val_loss: 0.7502948939800262 | val_accuracy: 72.43%
[19, 50] loss: 0.4886013180017471 | accuracy: 81.56% | val_loss: 0.6995030250400305 | val_accuracy: 75.22%
[19, 60] loss: 0.531373479962349 | accuracy: 78.75% | val_loss: 0.6821174351498485 | val_accuracy: 75.62%
[19, 70] loss: 0.5440706044435502 | accuracy: 80.31% | val_loss: 0.6965955151244998 | val_accuracy: 75.32%
[19, 80] loss: 0.48395558893680574 | accuracy: 83.44% | val_loss: 0.6783685479313135 | val_accuracy: 74.53%
[19, 90] loss: 0.4902141153812408 | accuracy: 80.62% | val_loss: 0.6702923877164721 | val_accuracy: 74.73%
[19, 100] loss: 0.47545807957649233 | accuracy: 83.75% | val_loss: 0.7165146072511561 | val_accuracy: 74.43%
[19, 110] loss: 0.46781986951828003 | accuracy: 83.75% | val_loss: 0.6631583878770471 | val_accuracy: 74.63%
[19, 120] loss: 0.4769197329878807 | accuracy: 83.12% | val_loss: 0.7348877303302288 | val_accuracy: 72.23%
[19, 130] loss: 0.5277956962585449 | accuracy: 80.31% | val_loss: 0.7261274200864136 | val_accuracy: 74.73%
[19, 140] loss: 0.4980957359075546 | accuracy: 82.81% | val_loss: 0.7745617171749473 | val_accuracy: 75.22%
[19, 150] loss: 0.5838368773460388 | accuracy: 76.25% | val_loss: 0.7034825701266527 | val_accuracy: 73.33%
[19, 160] loss: 0.5631841003894806 | accuracy: 81.56% | val_loss: 0.7905402733013034 | val_accuracy: 75.12%
[19, 170] loss: 0.5280616641044616 | accuracy: 80.31% | val_loss: 0.7506275428459048 | val_accuracy: 74.83%
[19, 180] loss: 0.5619235873222351 | accuracy: 79.06% | val_loss: 0.7063659923151135 | val_accuracy: 74.83%
[20, 10] loss: 0.46805990040302276 | accuracy: 83.44% | val_loss: 0.7157301530241966 | val_accuracy: 74.93%
[20, 20] loss: 0.5928396433591843 | accuracy: 77.50% | val_loss: 0.6807396290823817 | val_accuracy: 75.02%
[20, 30] loss: 0.46932328641414645 | accuracy: 82.50% | val_loss: 0.7149901231750846 | val_accuracy: 74.33%
[20, 40] loss: 0.3826304703950882 | accuracy: 85.62% | val_loss: 0.6916065057739615 | val_accuracy: 74.53%
[20, 50] loss: 0.43651013970375063 | accuracy: 82.81% | val_loss: 0.747936071595177 | val_accuracy: 75.02%
[20, 60] loss: 0.465143084526062 | accuracy: 84.38% | val_loss: 0.7074233908206224 | val_accuracy: 75.72%
[20, 70] loss: 0.47701441645622256 | accuracy: 81.88% | val_loss: 0.8409836292266846 | val_accuracy: 74.53%
[20, 80] loss: 0.47115504145622256 | accuracy: 81.25% | val_loss: 0.713282996788621 | val_accuracy: 75.42%
[20, 90] loss: 0.45367157012224196 | accuracy: 84.06% | val_loss: 0.707571784965694 | val_accuracy: 74.33%
[20, 100] loss: 0.5789996743202209 | accuracy: 75.94% | val_loss: 0.7057148739695549 | val_accuracy: 74.93%
[20, 110] loss: 0.5340734601020813 | accuracy: 78.75% | val_loss: 0.7190236747264862 | val_accuracy: 73.43%
[20, 120] loss: 0.5389476895332337 | accuracy: 79.38% | val_loss: 0.7094653332605958 | val_accuracy: 75.52%
[20, 130] loss: 0.48544315695762635 | accuracy: 82.19% | val_loss: 0.7067440040409565 | val_accuracy: 75.42%
[20, 140] loss: 0.4435017019510269 | accuracy: 85.00% | val_loss: 0.724676557816565 | val_accuracy: 76.52%
[20, 150] loss: 0.5068931370973587 | accuracy: 81.25% | val_loss: 0.7021793685853481 | val_accuracy: 75.32%
[20, 160] loss: 0.4611221671104431 | accuracy: 80.00% | val_loss: 0.7594579085707664 | val_accuracy: 74.23%
```

```
[20, 170] loss: 0.5360630750656128 | accuracy: 79.38% | val_loss: 0.8036945099011064 | val_accuracy: 74.73%
[20, 180] loss: 0.4569655045866966 | accuracy: 80.31% | val_loss: 0.7320058541372418 | val_accuracy: 72.93%
```

```
In [33]:  # Get the number_to_label_array so that we can use it later to plot the confusion matrix
          simple_number_to_label_array = testloader.dataset.dataset.number_to_label_array
          # Test the model
          test_net(model, testloader,simple_number_to_label_array, loss=loss, device=device)
```

```
Test loss: 0.7584205269813538 | Test accuracy: 73.74%
```
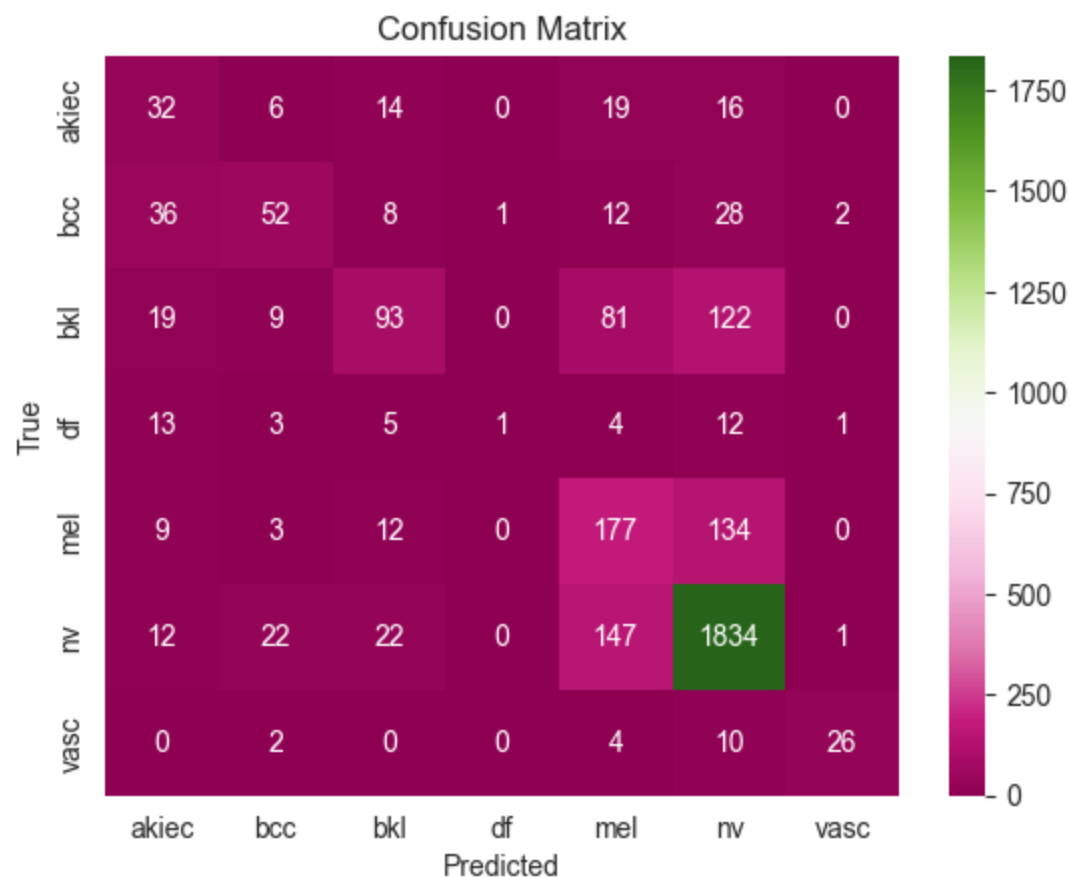

Observations

- We can see that the model has a strong inclination to predict the label `nv` and the label `mel` This might be because the dataset is imbalanced, as we can see bellow

- The model only predicts `df` once, which again is the least common label in the dataset
- Even though `vasc` is the second least common label, the model does a very good job at predicting it, both precision and recall are very good
- A lot of the predictions of `mel` are actually `nv`, which might mean that either the model is having a hard time distinguishing between the two, or that there are a lot of similarities between the two illnesses
- Even though `bkl` is the second most common label (close with `mel`), the model isn't very good at distinguishing it from `mel` and `nv`.
- We can also see that for `bkl` that has about the same number of samples as `mel`, that when the model was given a `bkl` image, that the predictions are more spread out, whereas for `mel` the predictions are more concentrated on `nv` and `mel`. Again this might strongly suggest that there are a lot of similarities between `mel` and `nv`
- The model's performance on the rest of the labels `akiec`, `bcc`, are very good (recall)

In [10]:
```python
label_counts = dataset.df['dx'].value_counts()
label_counts = [dataset.number_to_label_array[i] for i in label_counts.index]

plt.figure(figsize=(10,6))
plt.bar(label_counts, dataset.df['dx'].value_counts())
plt.show()
```
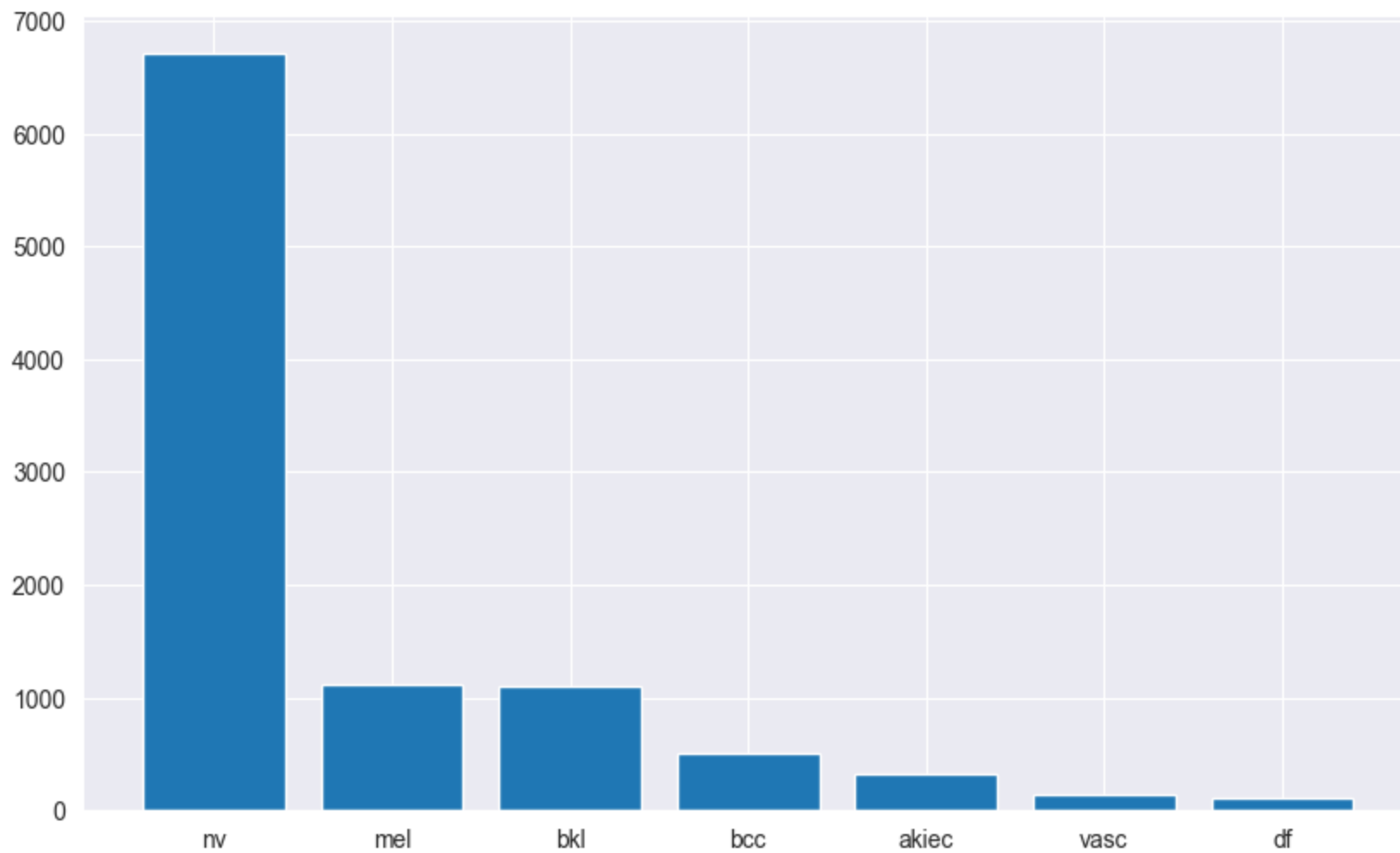
Now let's plot the loss and accuracy

In [11]:
```python
import copy
```

In [12]:
```python
# Copying the lists so that I don't overwrite them
simple_train_loss = copy.deepcopy(train_loss)
simple_val_loss = copy.deepcopy(val_loss)
simple_train_accuracy = copy.deepcopy(train_accuracy)
simple_val_accuracy = copy.deepcopy(val_accuracy)
```

In [13]:
```python
x_axis = list(map(lambda x:x*10,range(len(simple_train_loss))))
plt.figure(figsize=(10,6))
plt.plot(x_axis ,simple_train_loss, label='Training Loss')
plt.plot(x_axis , simple_val_loss,label='Validation Loss')
xticks = np.arange(x_axis[0], x_axis[-1]+1, 200.0)
plt.xticks([*xticks, x_axis[-1]])
plt.xlabel('Batch Number')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.grid()
plt.show()
```

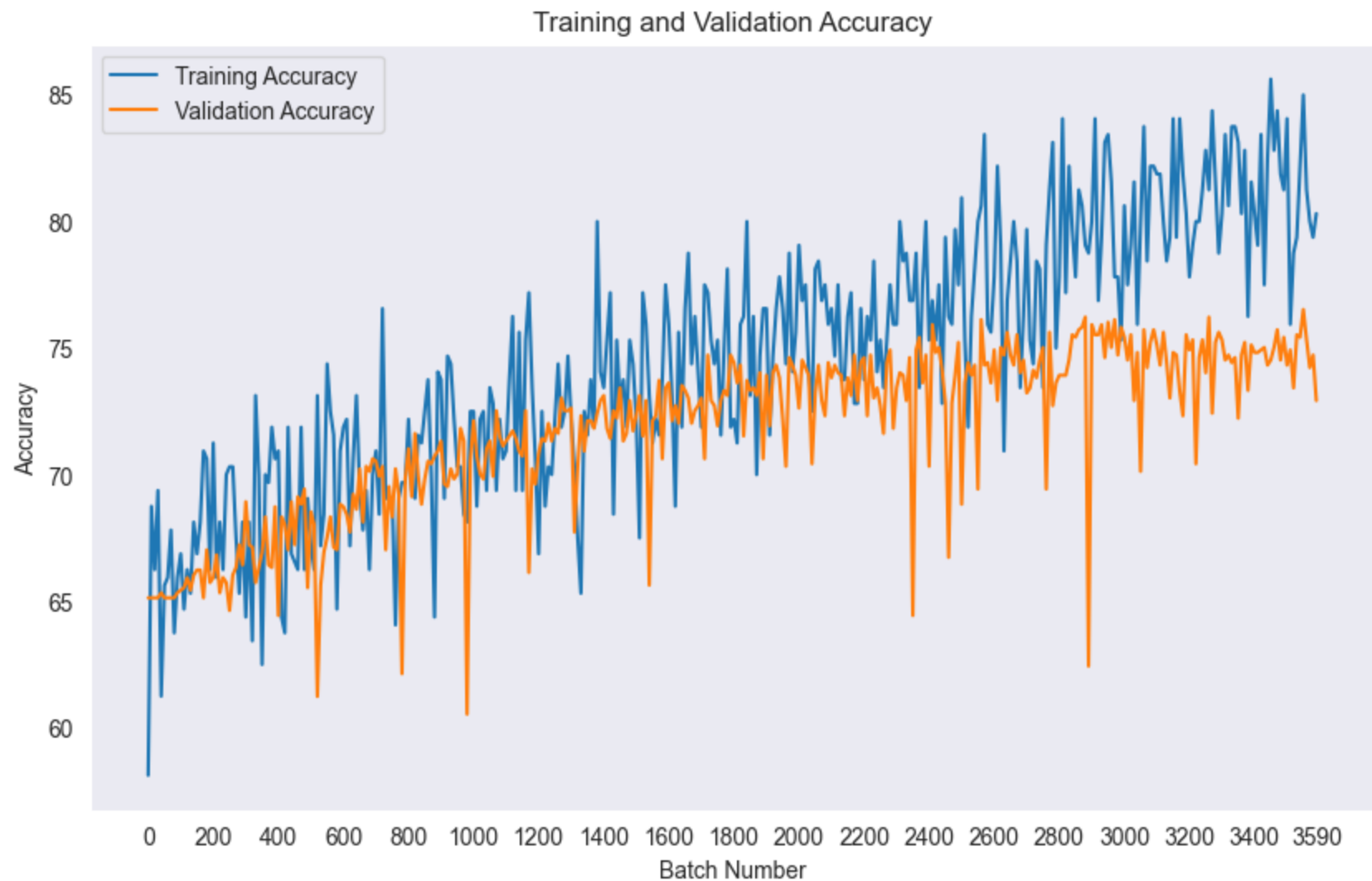Training and Validation Loss
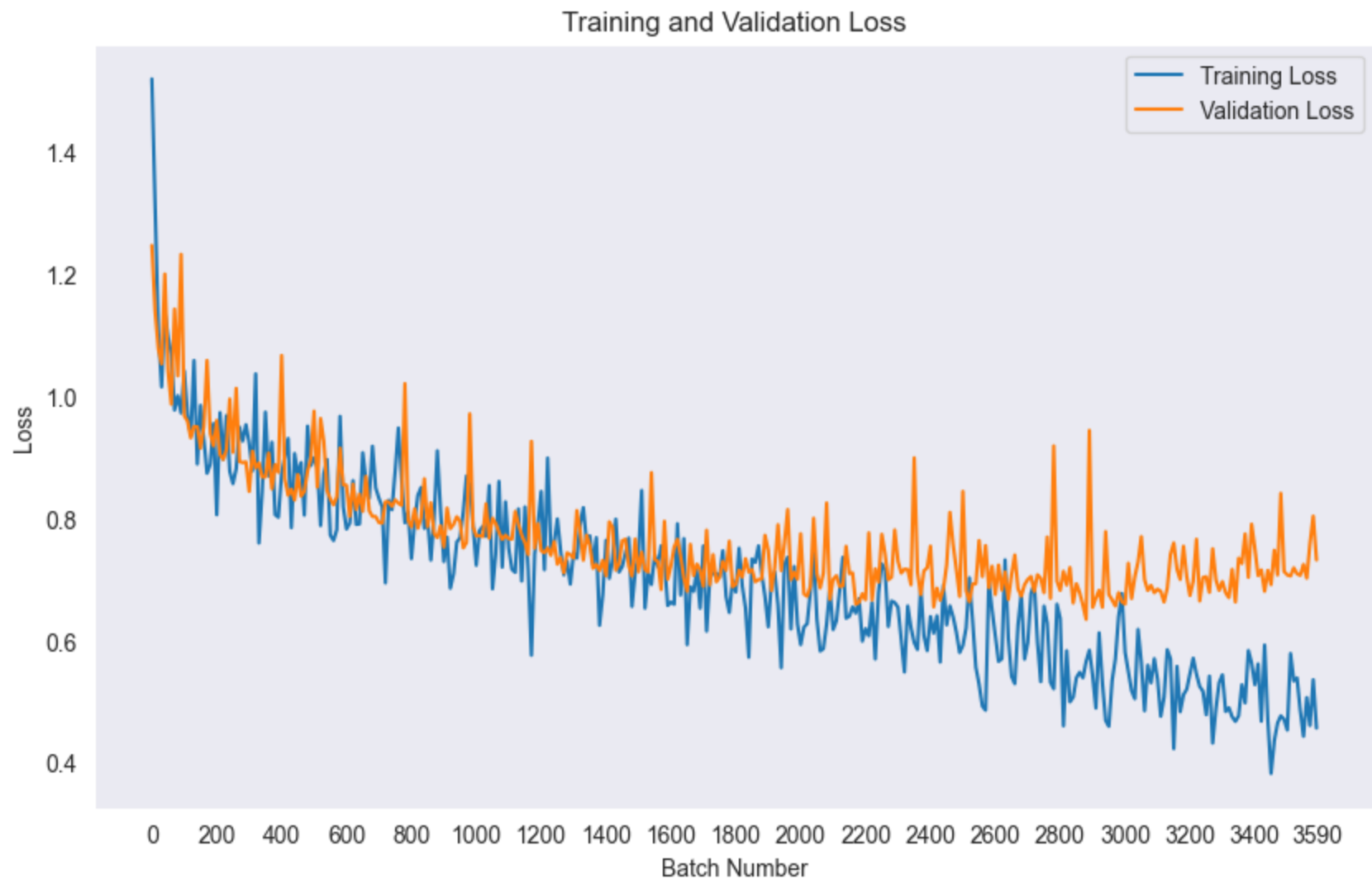
Observations

- From the loss plots I can conclude that the model might start to overfit at around 2500 batches, so we might want to stop the training at around that point
- The validation loss and the training loss are very close together, which is a good thing, because the while the model is improving on the training set, it is also improving on the validation set.

- A very negative observation is that both losses are fluctuating a lot, which might mean that the model is "overshooting" the minimum loss, therefore requiring more epochs to train

In [14]:
```python
x_axis = list(map(lambda x:x*10,range(len(simple_train_accuracy))))
plt.figure(figsize=(10,6))
plt.plot(x_axis ,simple_train_accuracy, label='Training Accuracy')
plt.plot(x_axis , simple_val_accuracy,label='Validation Accuracy')
xticks = np.arange(x_axis[0], x_axis[-1]+1, 200.0)
plt.xticks([*xticks, x_axis[-1]])
plt.xlabel('Batch Number')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.grid()
plt.show()
```

Training and Validation Accuracy

Observations

- From the accuracy plots we can still conclude that the model might start to overfit at around 2500 batches, which is good that both the loss and accuracy plots are showing the same thing
- The accuracy for both the training and validation set are increasing, which is a good thing, because the model is improving on both sets

- We can still see the big fluctuation we noticed on the loss plot before (negative).

Complex CNN

```
In [15]:  from torch.nn.functional import relu


          class ComplexConvNN(nn.Module):
              def __init__(self):
                  super(ComplexConvNN, self).__init__()
                  # In the specification of the assignment, it says that
                  # the images should be resized to [100 x 125] x 3 channels
                  # the comments bellow are the output image size after each layer
                  self.conv1 = nn.Conv2d(3, 32, 3) # 98 x 123 x 32
                  self.batchnorm1 = nn.BatchNorm2d(32) # 98 x 123 x 32
                  # MaxPool 2x2 49 x 61 x 32
                  self.conv2 = nn.Conv2d(32, 64, 3) # 47 x 59 x 64
                  self.batchnorm2 = nn.BatchNorm2d(64) # 47 x 59 x 64
                  # MaxPool 2x2 23 x 29 x 64
                  self.conv3 = nn.Conv2d(64, 128, 3) # 21 x 27 x 128
                  self.batchnorm3 = nn.BatchNorm2d(128) # 21 x 27 x 128
                  # MaxPool 2x2 10 x 13 x 128
                  self.conv4 = nn.Conv2d(128, 256, 3) # 8 x 11 x 256
                  self.batchnorm4 = nn.BatchNorm2d(256) # 8 x 11 x 256
                  # MaxPool 2x2 4 x 5 x 256
                  self.conv5 = nn.Conv2d(256, 512, 3) # 2 x 3 x 512
                  self.batchnorm5 = nn.BatchNorm2d(512) # 2 x 3 x 512
                  # MaxPool 2x2 1 x 1 x 512
                  self.fc = nn.Linear(512, 7)
                  self.maxpool = nn.MaxPool2d(2)
                  self.flatten = nn.Flatten()

              def forward(self, x):
                  # Ordering the layers as given in the assignment
                  x = self.maxpool(self.batchnorm1(relu(self.conv1(x))))
                  x = self.maxpool(self.batchnorm2(relu(self.conv2(x))))
                  x = self.maxpool(self.batchnorm3(relu(self.conv3(x))))
                  x = self.maxpool(self.batchnorm4(relu(self.conv4(x))))
                  x = self.maxpool(self.batchnorm5(relu(self.conv5(x))))
```

```
        x = nn.functional.adaptive_avg_pool2d(x,(1,1))
        x = self.flatten(x)
        x = self.fc(x)
        return x
```

Now let's train the Complex model and test it and see the difference

Note: naming everything complex_ so that I don't overwrite the simple model's variables

In [16]:
```python
from torchsummary import summary

# Constants from the assignment
complex_M = 100
complex_N = 125
# Create the transforms
complex_transforms = create_transform(complex_M, complex_N)
# Create the dataset
complex_dataset = MLProject2Dataset(data_dir='data', metadata_fname='metadata.csv', transform=complex_transforms)
# Split the dataset into train, validation and test
complex_train_dataset, complex_val_dataset, complex_test_dataset = torch.utils.data.random_split(
                                                        complex_dataset,
                                                        [.6, .1, .3],
                                                        generator=torch.Generator().manual_seed(42))

# Create the dataloaders
BATCH_SIZE = 32
complex_trainloader = DataLoader(complex_train_dataset, batch_size=BATCH_SIZE, shuffle=True)
complex_valloader = DataLoader(complex_val_dataset, batch_size=BATCH_SIZE, shuffle=True)
complex_testloader = DataLoader(complex_test_dataset, batch_size=BATCH_SIZE, shuffle=True)

# Create the model
complex_model = ComplexConvNN()
complex_model = complex_model.to(device)
# Print the model summary
summary(complex_model, (3, complex_M, complex_N))
# Create the optimizer
complex_optimizer = optim.Adam(complex_model.parameters(), lr=1e-3)
# Create the loss function
```

```python
complex_loss = nn.CrossEntropyLoss()
# Train the model
train_net(complex_model, complex_trainloader, complex_valloader, epochs=20, optimizer=complex_optimizer,
          loss=complex_loss, device=device)
# This cell took 44 minutes to run on my machine
```

```
----------------------------------------------------------------
        Layer (type)              Output Shape          Param #
================================================================
          Conv2d-1            [-1, 32, 98, 123]             896
     BatchNorm2d-2            [-1, 32, 98, 123]              64
       MaxPool2d-3             [-1, 32, 49, 61]               0
          Conv2d-4             [-1, 64, 47, 59]          18,496
     BatchNorm2d-5             [-1, 64, 47, 59]             128
       MaxPool2d-6             [-1, 64, 23, 29]               0
          Conv2d-7            [-1, 128, 21, 27]          73,856
     BatchNorm2d-8            [-1, 128, 21, 27]             256
       MaxPool2d-9            [-1, 128, 10, 13]               0
         Conv2d-10             [-1, 256, 8, 11]         295,168
    BatchNorm2d-11             [-1, 256, 8, 11]             512
      MaxPool2d-12              [-1, 256, 4, 5]               0
         Conv2d-13              [-1, 512, 2, 3]       1,180,160
    BatchNorm2d-14              [-1, 512, 2, 3]           1,024
      MaxPool2d-15              [-1, 512, 1, 1]               0
        Flatten-16                   [-1, 512]               0
         Linear-17                     [-1, 7]           3,591
================================================================
Total params: 1,574,151
Trainable params: 1,574,151
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.14
Forward/backward pass size (MB): 11.32
Params size (MB): 6.00
Estimated Total Size (MB): 17.47
----------------------------------------------------------------
[1, 10] loss: 1.525368857383728 | accuracy: 54.69% | val_loss: 1.754875309765339 | val_accuracy: 61.14%
[1, 20] loss: 1.048032408952713 | accuracy: 66.56% | val_loss: 1.4846876803785563 | val_accuracy: 66.03%
[1, 30] loss: 0.9679681122303009 | accuracy: 64.69% | val_loss: 1.3263956252485514 | val_accuracy: 64.34%
[1, 40] loss: 0.8475734531879425 | accuracy: 70.31% | val_loss: 1.196918735280633 | val_accuracy: 65.43%
[1, 50] loss: 0.7968601644039154 | accuracy: 71.56% | val_loss: 1.0745130609720945 | val_accuracy: 66.03%
[1, 60] loss: 0.925772750377655 | accuracy: 65.00% | val_loss: 0.9131527952849865 | val_accuracy: 67.23%
[1, 70] loss: 0.8625742673873902 | accuracy: 68.44% | val_loss: 0.9154977351427078 | val_accuracy: 68.03%
[1, 80] loss: 0.9956126093864441 | accuracy: 63.75% | val_loss: 0.8672284316271544 | val_accuracy: 69.13%
[1, 90] loss: 0.8503584682941436 | accuracy: 67.50% | val_loss: 0.9038623012602329 | val_accuracy: 67.73%
[1, 100] loss: 0.8787101924419403 | accuracy: 68.12% | val_loss: 0.8230524556711316 | val_accuracy: 68.83%
[1, 110] loss: 0.848471587896347 | accuracy: 69.06% | val_loss: 0.8648187052458525 | val_accuracy: 64.24%
[1, 120] loss: 0.846266633272171 | accuracy: 69.69% | val_loss: 0.9255834827199578 | val_accuracy: 71.03%
```

```
[1, 130] loss: 0.9086926519870758 | accuracy: 70.00% | val_loss: 0.9583376441150904 | val_accuracy: 61.64%
[1, 140] loss: 0.934523731470108 | accuracy: 67.19% | val_loss: 0.9434949308633804 | val_accuracy: 69.83%
[1, 150] loss: 0.9806777119636536 | accuracy: 64.69% | val_loss: 0.8877771589905024 | val_accuracy: 66.63%
[1, 160] loss: 0.9341613352298737 | accuracy: 66.25% | val_loss: 0.8553004846908152 | val_accuracy: 68.13%
[1, 170] loss: 0.7616441905498504 | accuracy: 74.06% | val_loss: 0.810623412951827 | val_accuracy: 70.63%
[1, 180] loss: 0.8601815044879914 | accuracy: 69.69% | val_loss: 0.7609581723809242 | val_accuracy: 70.13%
[2, 10] loss: 0.7945726901292801 | accuracy: 67.81% | val_loss: 0.7938999701291323 | val_accuracy: 71.23%
[2, 20] loss: 0.7408511936664581 | accuracy: 74.69% | val_loss: 0.8276758911088109 | val_accuracy: 69.23%
[2, 30] loss: 0.7311596751213074 | accuracy: 71.56% | val_loss: 0.7480264557525516 | val_accuracy: 72.33%
[2, 40] loss: 0.7770373463630676 | accuracy: 70.00% | val_loss: 0.8346471888944507 | val_accuracy: 67.83%
[2, 50] loss: 0.7937785059213638 | accuracy: 72.81% | val_loss: 0.8024060428142548 | val_accuracy: 71.83%
[2, 60] loss: 0.7525808155536652 | accuracy: 71.25% | val_loss: 0.8016196843236685 | val_accuracy: 71.23%
[2, 70] loss: 0.8220077574253082 | accuracy: 68.44% | val_loss: 0.7701370064169168 | val_accuracy: 73.13%
[2, 80] loss: 0.8187563627958298 | accuracy: 70.31% | val_loss: 0.8546833172440529 | val_accuracy: 70.13%
[2, 90] loss: 0.8573897421360016 | accuracy: 67.81% | val_loss: 0.7251671906560659 | val_accuracy: 71.03%
[2, 100] loss: 0.7590774774551392 | accuracy: 71.25% | val_loss: 0.736796161159873 | val_accuracy: 72.13%
[2, 110] loss: 0.7323598563671112 | accuracy: 72.81% | val_loss: 0.7469557916405797 | val_accuracy: 70.83%
[2, 120] loss: 0.7378456771373749 | accuracy: 74.38% | val_loss: 0.7975301695987582 | val_accuracy: 72.83%
[2, 130] loss: 0.7270847409963608 | accuracy: 73.12% | val_loss: 0.7631562286987901 | val_accuracy: 72.33%
[2, 140] loss: 0.8094313263893127 | accuracy: 71.56% | val_loss: 0.7409283518791199 | val_accuracy: 72.33%
[2, 150] loss: 0.7181434273719788 | accuracy: 73.12% | val_loss: 0.7025347780436277 | val_accuracy: 72.73%
[2, 160] loss: 0.7273870825767517 | accuracy: 73.12% | val_loss: 0.7537658223882318 | val_accuracy: 69.23%
[2, 170] loss: 0.7126692235469818 | accuracy: 74.06% | val_loss: 0.8142985589802265 | val_accuracy: 70.23%
[2, 180] loss: 0.6493846714496613 | accuracy: 76.25% | val_loss: 0.7875482980161905 | val_accuracy: 72.03%
[3, 10] loss: 0.7981949210166931 | accuracy: 72.50% | val_loss: 0.7243238845840096 | val_accuracy: 73.33%
[3, 20] loss: 0.7865974605083466 | accuracy: 70.31% | val_loss: 0.7377853617072105 | val_accuracy: 72.33%
[3, 30] loss: 0.7596865117549896 | accuracy: 70.31% | val_loss: 0.7134949490427971 | val_accuracy: 73.13%
[3, 40] loss: 0.6869500339031219 | accuracy: 75.94% | val_loss: 0.7359898155555129 | val_accuracy: 72.63%
[3, 50] loss: 0.6931857079267502 | accuracy: 74.06% | val_loss: 0.6665789466351271 | val_accuracy: 75.82%
[3, 60] loss: 0.6467795431613922 | accuracy: 74.38% | val_loss: 0.6891673523932695 | val_accuracy: 73.73%
[3, 70] loss: 0.7569205075502395 | accuracy: 69.69% | val_loss: 0.7441888442263007 | val_accuracy: 71.93%
[3, 80] loss: 0.5871208906173706 | accuracy: 79.06% | val_loss: 0.755314220674336 | val_accuracy: 72.63%
[3, 90] loss: 0.6852631628513336 | accuracy: 73.75% | val_loss: 0.69936529360711157 | val_accuracy: 72.93%
[3, 100] loss: 0.6780262380838394 | accuracy: 73.44% | val_loss: 0.718998109921813 | val_accuracy: 73.63%
[3, 110] loss: 0.6798575162887573 | accuracy: 73.12% | val_loss: 0.6961178304627538 | val_accuracy: 74.33%
[3, 120] loss: 0.640699279308319 | accuracy: 75.62% | val_loss: 0.8220480047166348 | val_accuracy: 71.23%
[3, 130] loss: 0.660409563779831 | accuracy: 75.94% | val_loss: 0.764733461663127 | val_accuracy: 71.83%
[3, 140] loss: 0.7896904170513153 | accuracy: 70.31% | val_loss: 0.7851889301091433 | val_accuracy: 72.13%
[3, 150] loss: 0.7337817490100861 | accuracy: 72.50% | val_loss: 0.7321496540680528 | val_accuracy: 73.33%
[3, 160] loss: 0.762595921754837 | accuracy: 73.44% | val_loss: 0.7484336290508509 | val_accuracy: 71.33%
[3, 170] loss: 0.6975354492664337 | accuracy: 76.56% | val_loss: 0.818058161996305 | val_accuracy: 70.03%
[3, 180] loss: 0.7101779103279113 | accuracy: 74.06% | val_loss: 0.7126108082011342 | val_accuracy: 72.53%
```

```
[4, 10] loss: 0.6851392537355423 | accuracy: 73.12% | val_loss: 0.6661160225048661 | val_accuracy: 74.53%
[4, 20] loss: 0.6143472164869308 | accuracy: 77.50% | val_loss: 0.7308760536834598 | val_accuracy: 71.33%
[4, 30] loss: 0.6243782639503479 | accuracy: 77.19% | val_loss: 0.7943532140925527 | val_accuracy: 72.93%
[4, 40] loss: 0.6495967596769333 | accuracy: 75.31% | val_loss: 0.690502280369401 | val_accuracy: 72.83%
[4, 50] loss: 0.6017820715904236 | accuracy: 77.19% | val_loss: 0.7567709516733885 | val_accuracy: 72.63%
[4, 60] loss: 0.6245465099811554 | accuracy: 78.44% | val_loss: 0.6946321725845337 | val_accuracy: 73.73%
[4, 70] loss: 0.5639015108346939 | accuracy: 79.69% | val_loss: 0.7111127162352204 | val_accuracy: 73.53%
[4, 80] loss: 0.6008388489484787 | accuracy: 79.69% | val_loss: 0.7157345283776522 | val_accuracy: 73.83%
[4, 90] loss: 0.6503967195749283 | accuracy: 74.69% | val_loss: 0.7042385125532746 | val_accuracy: 74.03%
[4, 100] loss: 0.7284781575202942 | accuracy: 72.50% | val_loss: 0.7170220529660583 | val_accuracy: 73.03%
[4, 110] loss: 0.6417625278234482 | accuracy: 75.62% | val_loss: 0.7443004930391908 | val_accuracy: 72.23%
[4, 120] loss: 0.7383839577436447 | accuracy: 73.44% | val_loss: 0.730309440754354 | val_accuracy: 70.93%
[4, 130] loss: 0.6641679525375366 | accuracy: 73.44% | val_loss: 0.661432571709156 | val_accuracy: 75.02%
[4, 140] loss: 0.5629128009080887 | accuracy: 81.25% | val_loss: 0.6822343049570918 | val_accuracy: 72.93%
[4, 150] loss: 0.5912684619426727 | accuracy: 78.44% | val_loss: 0.6753487708047032 | val_accuracy: 74.03%
[4, 160] loss: 0.6912270188331604 | accuracy: 76.25% | val_loss: 0.7066282192245126 | val_accuracy: 74.03%
[4, 170] loss: 0.6448119044303894 | accuracy: 78.44% | val_loss: 0.7190633974969387 | val_accuracy: 74.23%
[4, 180] loss: 0.6659507751464844 | accuracy: 75.62% | val_loss: 0.7199621144682169 | val_accuracy: 72.63%
[5, 10] loss: 0.42747653275728226 | accuracy: 84.38% | val_loss: 0.7026381734758615 | val_accuracy: 74.53%
[5, 20] loss: 0.628812262415886 | accuracy: 77.19% | val_loss: 0.6493992051109672 | val_accuracy: 75.52%
[5, 30] loss: 0.6150345742702484 | accuracy: 74.69% | val_loss: 0.752644918859005 | val_accuracy: 73.63%
[5, 40] loss: 0.5681629598140716 | accuracy: 80.94% | val_loss: 0.6697120564058423 | val_accuracy: 74.93%
[5, 50] loss: 0.6551063001155853 | accuracy: 72.81% | val_loss: 0.6395934196189046 | val_accuracy: 76.22%
[5, 60] loss: 0.5988477528095245 | accuracy: 77.81% | val_loss: 0.760942105203867 | val_accuracy: 73.53%
[5, 70] loss: 0.6526502728462219 | accuracy: 75.00% | val_loss: 0.7256879229098558 | val_accuracy: 73.03%
[5, 80] loss: 0.5281339973211289 | accuracy: 80.00% | val_loss: 0.7300684247165918 | val_accuracy: 73.53%
[5, 90] loss: 0.5579124093055725 | accuracy: 80.31% | val_loss: 0.6426172945648432 | val_accuracy: 76.82%
[5, 100] loss: 0.6096403241157532 | accuracy: 79.38% | val_loss: 0.7014140486717224 | val_accuracy: 73.83%
[5, 110] loss: 0.6089135110378265 | accuracy: 77.19% | val_loss: 0.6939684972167015 | val_accuracy: 72.93%
[5, 120] loss: 0.5378909409046173 | accuracy: 79.06% | val_loss: 0.6781051466241479 | val_accuracy: 74.33%
[5, 130] loss: 0.6228233128786087 | accuracy: 77.19% | val_loss: 0.6672952435910702 | val_accuracy: 74.93%
[5, 140] loss: 0.6258638113737106 | accuracy: 77.50% | val_loss: 0.701608058065176 | val_accuracy: 74.33%
[5, 150] loss: 0.5597929537296296 | accuracy: 80.00% | val_loss: 0.742314176633954 | val_accuracy: 73.43%
[5, 160] loss: 0.602263480424881 | accuracy: 78.12% | val_loss: 0.6921169459819794 | val_accuracy: 74.43%
[5, 170] loss: 0.6528618723154068 | accuracy: 76.88% | val_loss: 0.7696810476481915 | val_accuracy: 71.83%
[5, 180] loss: 0.6011953115463257 | accuracy: 77.19% | val_loss: 0.7547702193260193 | val_accuracy: 74.23%
[6, 10] loss: 0.5242511987686157 | accuracy: 82.19% | val_loss: 0.7555083157494664 | val_accuracy: 73.13%
[6, 20] loss: 0.531225460767746 | accuracy: 80.62% | val_loss: 0.7659792574122548 | val_accuracy: 74.03%
[6, 30] loss: 0.5498735547065735 | accuracy: 80.31% | val_loss: 0.7253838591277599 | val_accuracy: 74.03%
[6, 40] loss: 0.5838291585445404 | accuracy: 76.56% | val_loss: 0.7196865091100335 | val_accuracy: 74.63%
[6, 50] loss: 0.5515437334775924 | accuracy: 81.56% | val_loss: 0.6670173043385148 | val_accuracy: 74.73%
[6, 60] loss: 0.5277738034725189 | accuracy: 80.94% | val_loss: 0.6623294213786721 | val_accuracy: 75.32%
```

```
[6, 70] loss: 0.5436754882335663 | accuracy: 79.69% | val_loss: 0.6594334347173572 | val_accuracy: 75.92%
[6, 80] loss: 0.5011586725711823 | accuracy: 83.44% | val_loss: 0.6716408152133226 | val_accuracy: 74.83%
[6, 90] loss: 0.44178597033023836 | accuracy: 83.44% | val_loss: 0.7686292761936784 | val_accuracy: 73.43%
[6, 100] loss: 0.572308573126793 | accuracy: 77.81% | val_loss: 0.6937563940882683 | val_accuracy: 75.82%
[6, 110] loss: 0.5898539304733277 | accuracy: 78.75% | val_loss: 0.762327391654253 | val_accuracy: 71.43%
[6, 120] loss: 0.6022080183029175 | accuracy: 76.25% | val_loss: 0.7577684153802693 | val_accuracy: 74.23%
[6, 130] loss: 0.6230683565139771 | accuracy: 77.81% | val_loss: 0.7852026764303446 | val_accuracy: 70.93%
[6, 140] loss: 0.6845522135496139 | accuracy: 74.38% | val_loss: 0.7362423706799746 | val_accuracy: 73.63%
[6, 150] loss: 0.5716134905815125 | accuracy: 80.94% | val_loss: 0.7635836079716682 | val_accuracy: 74.93%
[6, 160] loss: 0.6003013968467712 | accuracy: 79.06% | val_loss: 0.6981897838413715 | val_accuracy: 74.93%
[6, 170] loss: 0.618625384569168 | accuracy: 78.12% | val_loss: 0.7180526209995151 | val_accuracy: 73.23%
[6, 180] loss: 0.5887133687734604 | accuracy: 77.81% | val_loss: 0.70682492852211 | val_accuracy: 73.93%
[7, 10] loss: 0.5634248942136765 | accuracy: 76.56% | val_loss: 0.6966182123869658 | val_accuracy: 74.83%
[7, 20] loss: 0.5357971429824829 | accuracy: 78.75% | val_loss: 0.7289251117035747 | val_accuracy: 75.22%
[7, 30] loss: 0.4663150429725647 | accuracy: 84.38% | val_loss: 0.6549995937384665 | val_accuracy: 74.03%
[7, 40] loss: 0.5001370221376419 | accuracy: 79.38% | val_loss: 0.6826645396649837 | val_accuracy: 76.42%
[7, 50] loss: 0.45196916460990905 | accuracy: 83.75% | val_loss: 0.6773256994783878 | val_accuracy: 76.52%
[7, 60] loss: 0.5751553356647492 | accuracy: 79.06% | val_loss: 0.6490303492173553 | val_accuracy: 75.12%
[7, 70] loss: 0.4658581018447876 | accuracy: 82.50% | val_loss: 0.6900851363316178 | val_accuracy: 76.12%
[7, 80] loss: 0.5201054662466049 | accuracy: 80.31% | val_loss: 0.8181768758222461 | val_accuracy: 69.73%
[7, 90] loss: 0.49066494405269623 | accuracy: 81.25% | val_loss: 0.8877650555223227 | val_accuracy: 71.93%
[7, 100] loss: 0.4751939564943314 | accuracy: 83.12% | val_loss: 0.7934413435868919 | val_accuracy: 73.03%
[7, 110] loss: 0.47964825928211213 | accuracy: 80.94% | val_loss: 0.7247113166376948 | val_accuracy: 74.53%
[7, 120] loss: 0.5779032170772552 | accuracy: 79.06% | val_loss: 0.7125228801742196 | val_accuracy: 74.03%
[7, 130] loss: 0.5329542577266693 | accuracy: 80.94% | val_loss: 0.756137597374618 | val_accuracy: 73.53%
[7, 140] loss: 0.5919827550649643 | accuracy: 78.44% | val_loss: 0.669350235722959 | val_accuracy: 75.32%
[7, 150] loss: 0.5527170091867447 | accuracy: 78.44% | val_loss: 0.6826230697333813 | val_accuracy: 75.72%
[7, 160] loss: 0.5015882283449173 | accuracy: 78.44% | val_loss: 0.7122407080605626 | val_accuracy: 72.93%
[7, 170] loss: 0.4937757521867752 | accuracy: 82.19% | val_loss: 0.7171920370310545 | val_accuracy: 74.83%
[7, 180] loss: 0.5859994709491729 | accuracy: 77.19% | val_loss: 0.7165569718927145 | val_accuracy: 73.03%
[8, 10] loss: 0.4640814781188965 | accuracy: 83.75% | val_loss: 0.6844212589785457 | val_accuracy: 75.82%
[8, 20] loss: 0.4802232250571251 | accuracy: 84.06% | val_loss: 0.6653440967202187 | val_accuracy: 76.02%
[8, 30] loss: 0.346993026137352 | accuracy: 88.44% | val_loss: 0.7225746782496572 | val_accuracy: 74.53%
[8, 40] loss: 0.42597312331199644 | accuracy: 85.00% | val_loss: 0.7238705363124609 | val_accuracy: 73.43%
[8, 50] loss: 0.46203380823135376 | accuracy: 84.38% | val_loss: 0.7357659935951233 | val_accuracy: 75.72%
[8, 60] loss: 0.4289455056190491 | accuracy: 83.75% | val_loss: 0.7429496459662914 | val_accuracy: 76.32%
[8, 70] loss: 0.4391680970788002 | accuracy: 85.00% | val_loss: 0.703657578676939 | val_accuracy: 74.93%
[8, 80] loss: 0.4891479820013046 | accuracy: 80.62% | val_loss: 0.7675921767950058 | val_accuracy: 72.53%
[8, 90] loss: 0.42904698848724365 | accuracy: 80.94% | val_loss: 0.7237844932824373 | val_accuracy: 75.72%
[8, 100] loss: 0.49800845682621003 | accuracy: 81.25% | val_loss: 0.7427102653309703 | val_accuracy: 75.62%
[8, 110] loss: 0.4144706666469574 | accuracy: 83.12% | val_loss: 0.6443719221279025 | val_accuracy: 76.42%
[8, 120] loss: 0.44645904898643496 | accuracy: 84.06% | val_loss: 0.7195924250409007 | val_accuracy: 76.32%
```

```
[8, 130] loss: 0.41813180297613145 | accuracy: 83.12% | val_loss: 0.6679299352690578 | val_accuracy: 76.62%
[8, 140] loss: 0.4510944664478302 | accuracy: 83.44% | val_loss: 0.7720366399735212 | val_accuracy: 74.63%
[8, 150] loss: 0.43860664069652555 | accuracy: 83.12% | val_loss: 0.7692750254645944 | val_accuracy: 74.03%
[8, 160] loss: 0.460187965631485 | accuracy: 84.38% | val_loss: 0.8037289464846253 | val_accuracy: 72.43%
[8, 170] loss: 0.5126987636089325 | accuracy: 80.00% | val_loss: 0.7359307929873466 | val_accuracy: 74.33%
[8, 180] loss: 0.4864861384034157 | accuracy: 80.62% | val_loss: 0.7266455860808492 | val_accuracy: 75.02%
[9, 10] loss: 0.3166169703006744 | accuracy: 86.56% | val_loss: 0.7393769845366478 | val_accuracy: 75.42%
[9, 20] loss: 0.384126278758049 | accuracy: 88.44% | val_loss: 0.6952272485941648 | val_accuracy: 75.12%
[9, 30] loss: 0.34936831295490267 | accuracy: 84.38% | val_loss: 0.7418244276195765 | val_accuracy: 74.03%
[9, 40] loss: 0.3787523075938225 | accuracy: 86.56% | val_loss: 0.7207141928374767 | val_accuracy: 74.73%
[9, 50] loss: 0.3377505086362362 | accuracy: 85.31% | val_loss: 0.7638653377071023 | val_accuracy: 75.12%
[9, 60] loss: 0.39897020906209946 | accuracy: 85.94% | val_loss: 0.7349828770384192 | val_accuracy: 75.42%
[9, 70] loss: 0.4033952057361603 | accuracy: 84.38% | val_loss: 0.7435270177666098 | val_accuracy: 75.02%
[9, 80] loss: 0.3554788395762444 | accuracy: 86.56% | val_loss: 0.7502064639702439 | val_accuracy: 74.33%
[9, 90] loss: 0.32988495901227 | accuracy: 88.44% | val_loss: 0.7130730152130127 | val_accuracy: 76.92%
[9, 100] loss: 0.33826935589313506 | accuracy: 86.56% | val_loss: 0.6990932701155543 | val_accuracy: 74.73%
[9, 110] loss: 0.3701410606503487 | accuracy: 86.25% | val_loss: 0.7560901842080057 | val_accuracy: 75.82%
[9, 120] loss: 0.3612973004579544 | accuracy: 86.25% | val_loss: 0.7474024202674627 | val_accuracy: 71.53%
[9, 130] loss: 0.425904244184494 | accuracy: 85.00% | val_loss: 0.8258399926126003 | val_accuracy: 72.33%
[9, 140] loss: 0.41219498217105865 | accuracy: 85.62% | val_loss: 0.7315480476245284 | val_accuracy: 75.12%
[9, 150] loss: 0.46191360503435136 | accuracy: 81.88% | val_loss: 0.8383160904049873 | val_accuracy: 75.62%
[9, 160] loss: 0.4874177575111389 | accuracy: 85.31% | val_loss: 0.7688255310058594 | val_accuracy: 75.12%
[9, 170] loss: 0.48077191412448883 | accuracy: 80.94% | val_loss: 0.7402738416567445 | val_accuracy: 73.93%
[9, 180] loss: 0.4294442892074585 | accuracy: 82.50% | val_loss: 0.8797552864998579 | val_accuracy: 72.53%
[10, 10] loss: 0.32960854172706605 | accuracy: 89.38% | val_loss: 0.7019726652652025 | val_accuracy: 76.42%
[10, 20] loss: 0.230664724111557 | accuracy: 93.12% | val_loss: 0.6882468909025192 | val_accuracy: 76.92%
[10, 30] loss: 0.2841650083661079 | accuracy: 89.38% | val_loss: 0.676266735419631 | val_accuracy: 75.52%
[10, 40] loss: 0.2350860580801964 | accuracy: 93.12% | val_loss: 0.7379265613853931 | val_accuracy: 75.32%
[10, 50] loss: 0.2489873692393303 | accuracy: 90.94% | val_loss: 0.6906312704086304 | val_accuracy: 77.02%
[10, 60] loss: 0.2746861234307289 | accuracy: 88.75% | val_loss: 0.7260429067537189 | val_accuracy: 76.12%
[10, 70] loss: 0.25803551375865935 | accuracy: 90.00% | val_loss: 0.7900746334344149 | val_accuracy: 75.72%
[10, 80] loss: 0.3102178543806076 | accuracy: 87.81% | val_loss: 0.7321698535233736 | val_accuracy: 73.93%
[10, 90] loss: 0.4026780426502228 | accuracy: 85.00% | val_loss: 0.8837671685032547 | val_accuracy: 73.03%
[10, 100] loss: 0.42276095151901244 | accuracy: 86.56% | val_loss: 0.8278008848428726 | val_accuracy: 72.23%
[10, 110] loss: 0.322435200214386 | accuracy: 86.88% | val_loss: 0.8874377887696028 | val_accuracy: 72.73%
[10, 120] loss: 0.28960962146520614 | accuracy: 88.75% | val_loss: 0.7740703811869025 | val_accuracy: 75.92%
[10, 130] loss: 0.2674565531313419 | accuracy: 90.62% | val_loss: 0.7284584818407893 | val_accuracy: 76.32%
[10, 140] loss: 0.27849512100219725 | accuracy: 90.94% | val_loss: 0.7557185404002666 | val_accuracy: 77.32%
[10, 150] loss: 0.2491996854543686 | accuracy: 90.31% | val_loss: 0.7652645688503981 | val_accuracy: 75.82%
[10, 160] loss: 0.28879406601190566 | accuracy: 90.62% | val_loss: 0.7480279952287674 | val_accuracy: 77.32%
[10, 170] loss: 0.304297436773777 | accuracy: 88.75% | val_loss: 0.729223994538188 | val_accuracy: 76.72%
[10, 180] loss: 0.4278077632188797 | accuracy: 83.75% | val_loss: 0.8021510299295187 | val_accuracy: 73.33%
```

```
[11, 10] loss: 0.20485142916440963 | accuracy: 92.19% | val_loss: 0.8090666546486318 | val_accuracy: 73.43%
[11, 20] loss: 0.22460691630840302 | accuracy: 90.94% | val_loss: 0.8081153770908713 | val_accuracy: 74.63%
[11, 30] loss: 0.18803773447871208 | accuracy: 93.75% | val_loss: 0.7663395954295993 | val_accuracy: 77.12%
[11, 40] loss: 0.18693647906184196 | accuracy: 93.44% | val_loss: 0.8091229954734445 | val_accuracy: 75.32%
[11, 50] loss: 0.3013253159821033 | accuracy: 87.19% | val_loss: 1.0432086903601885 | val_accuracy: 71.63%
[11, 60] loss: 0.2305863991379738 | accuracy: 92.19% | val_loss: 0.7753614038228989 | val_accuracy: 76.12%
[11, 70] loss: 0.4126579537987709 | accuracy: 86.25% | val_loss: 0.8146556178107858 | val_accuracy: 75.92%
[11, 80] loss: 0.303260849416256 | accuracy: 88.75% | val_loss: 0.8859371379949152 | val_accuracy: 73.23%
[11, 90] loss: 0.285899792611599 | accuracy: 88.12% | val_loss: 0.8209557188674808 | val_accuracy: 74.13%
[11, 100] loss: 0.32080396115779874 | accuracy: 86.88% | val_loss: 0.80959093850106 | val_accuracy: 74.43%
[11, 110] loss: 0.3036200061440468 | accuracy: 88.75% | val_loss: 0.7645796267315745 | val_accuracy: 76.42%
[11, 120] loss: 0.3117756336927414 | accuracy: 88.75% | val_loss: 0.7444771328009665 | val_accuracy: 75.92%
[11, 130] loss: 0.3985989898443222 | accuracy: 85.94% | val_loss: 0.7414070246741176 | val_accuracy: 74.93%
[11, 140] loss: 0.35424110740423204 | accuracy: 84.69% | val_loss: 0.7464679926633835 | val_accuracy: 75.92%
[11, 150] loss: 0.2665899500250816 | accuracy: 90.62% | val_loss: 0.8216437450610101 | val_accuracy: 75.22%
[11, 160] loss: 0.2902370892465115 | accuracy: 91.25% | val_loss: 0.789897358044982 | val_accuracy: 74.53%
[11, 170] loss: 0.23488557934761048 | accuracy: 91.25% | val_loss: 0.8302321615628898 | val_accuracy: 74.73%
[11, 180] loss: 0.25668537318706514 | accuracy: 92.19% | val_loss: 0.898239653557539 | val_accuracy: 73.43%
[12, 10] loss: 0.17189072966575622 | accuracy: 94.06% | val_loss: 0.7299645398743451 | val_accuracy: 76.92%
[12, 20] loss: 0.18843016475439073 | accuracy: 93.44% | val_loss: 0.8331656884402037 | val_accuracy: 75.12%
[12, 30] loss: 0.18653838634490966 | accuracy: 91.88% | val_loss: 0.7995128836482763 | val_accuracy: 75.52%
[12, 40] loss: 0.14363089874386786 | accuracy: 96.25% | val_loss: 0.7840799847617745 | val_accuracy: 76.02%
[12, 50] loss: 0.17881478667259215 | accuracy: 93.12% | val_loss: 0.8264884324744344 | val_accuracy: 77.02%
[12, 60] loss: 0.18179772570729255 | accuracy: 94.06% | val_loss: 0.8232440818101168 | val_accuracy: 75.42%
[12, 70] loss: 0.16494905091822148 | accuracy: 95.31% | val_loss: 0.8706642417237163 | val_accuracy: 75.12%
[12, 80] loss: 0.17275362685322762 | accuracy: 94.38% | val_loss: 0.8301319209858775 | val_accuracy: 74.53%
[12, 90] loss: 0.1899953603744507 | accuracy: 93.75% | val_loss: 0.8244237517938018 | val_accuracy: 74.53%
[12, 100] loss: 0.17877879813313485 | accuracy: 93.12% | val_loss: 0.8401510482653975 | val_accuracy: 75.92%
[12, 110] loss: 0.19057614281773566 | accuracy: 93.44% | val_loss: 0.8433142928406596 | val_accuracy: 77.42%
[12, 120] loss: 0.12587017845362425 | accuracy: 95.62% | val_loss: 0.8734995201230049 | val_accuracy: 75.42%
[12, 130] loss: 0.2095507375895977 | accuracy: 92.81% | val_loss: 0.8544324692338705 | val_accuracy: 77.12%
[12, 140] loss: 0.19610246866941453 | accuracy: 92.19% | val_loss: 0.8447294514626265 | val_accuracy: 74.53%
[12, 150] loss: 0.1994698651134968 | accuracy: 93.12% | val_loss: 0.8893896332010627 | val_accuracy: 75.82%
[12, 160] loss: 0.2374863713979721 | accuracy: 92.50% | val_loss: 0.9530644482001662 | val_accuracy: 74.53%
[12, 170] loss: 0.18425684422254562 | accuracy: 93.12% | val_loss: 0.8772980635985732 | val_accuracy: 76.42%
[12, 180] loss: 0.22585366144776345 | accuracy: 91.88% | val_loss: 0.8799355593509972 | val_accuracy: 76.62%
[13, 10] loss: 0.11684320718050004 | accuracy: 94.69% | val_loss: 0.9171552592888474 | val_accuracy: 75.42%
[13, 20] loss: 0.09838083013892174 | accuracy: 97.19% | val_loss: 0.8840879998169839 | val_accuracy: 75.42%
[13, 30] loss: 0.1121705312281847 | accuracy: 95.94% | val_loss: 0.9080717815086246 | val_accuracy: 76.02%
[13, 40] loss: 0.07448841854929925 | accuracy: 96.88% | val_loss: 0.8930431200424209 | val_accuracy: 76.22%
[13, 50] loss: 0.08496476169675589 | accuracy: 97.19% | val_loss: 0.8916977224871516 | val_accuracy: 75.52%
[13, 60] loss: 0.10428302772343159 | accuracy: 95.62% | val_loss: 0.9002576246857643 | val_accuracy: 76.82%
```

```
[13, 70] loss: 0.09401681665331126 | accuracy: 97.19% | val_loss: 0.930202744435519 | val_accuracy: 76.92%
[13, 80] loss: 0.11778297163546085 | accuracy: 95.94% | val_loss: 0.8923173239454627 | val_accuracy: 75.02%
[13, 90] loss: 0.17582507357001304 | accuracy: 92.50% | val_loss: 0.9215852450579405 | val_accuracy: 76.22%
[13, 100] loss: 0.09754376448690891 | accuracy: 96.56% | val_loss: 0.9387229764834046 | val_accuracy: 76.22%
[13, 110] loss: 0.14602591842412949 | accuracy: 95.94% | val_loss: 0.9933974640443921 | val_accuracy: 76.02%
[13, 120] loss: 0.2238267980515957 | accuracy: 92.19% | val_loss: 0.971290459856391 | val_accuracy: 74.03%
[13, 130] loss: 0.22838102877140046 | accuracy: 89.69% | val_loss: 0.9918082654476166 | val_accuracy: 76.42%
[13, 140] loss: 0.15112231075763702 | accuracy: 94.38% | val_loss: 0.8657059650868177 | val_accuracy: 74.43%
[13, 150] loss: 0.14471314158290624 | accuracy: 94.38% | val_loss: 0.8998843785375357 | val_accuracy: 76.12%
[13, 160] loss: 0.17268291488289833 | accuracy: 92.81% | val_loss: 1.045113236643374 | val_accuracy: 75.42%
[13, 170] loss: 0.1448018416762352 | accuracy: 95.00% | val_loss: 0.9915933618322015 | val_accuracy: 74.23%
[13, 180] loss: 0.15660530161112546 | accuracy: 95.00% | val_loss: 0.9095904808491468 | val_accuracy: 75.22%
[14, 10] loss: 0.1092452647164464 | accuracy: 96.88% | val_loss: 0.9795179031789303 | val_accuracy: 75.42%
[14, 20] loss: 0.11364899054169655 | accuracy: 96.56% | val_loss: 0.879524442832917 | val_accuracy: 75.22%
[14, 30] loss: 0.09839461799710988 | accuracy: 96.25% | val_loss: 0.86506960215047 | val_accuracy: 76.22%
[14, 40] loss: 0.08732141219079495 | accuracy: 96.88% | val_loss: 0.8699559001252055 | val_accuracy: 77.02%
[14, 50] loss: 0.11512785851955414 | accuracy: 96.56% | val_loss: 0.9614036418497562 | val_accuracy: 74.93%
[14, 60] loss: 0.07953784661367536 | accuracy: 98.12% | val_loss: 1.0151421763002872 | val_accuracy: 75.02%
[14, 70] loss: 0.13037337847054004 | accuracy: 95.94% | val_loss: 0.9893855499103665 | val_accuracy: 74.23%
[14, 80] loss: 0.07448555342853069 | accuracy: 97.81% | val_loss: 1.0185956303030252 | val_accuracy: 76.22%
[14, 90] loss: 0.08787544537335634 | accuracy: 97.81% | val_loss: 0.9538600156083703 | val_accuracy: 77.12%
[14, 100] loss: 0.07245448045432568 | accuracy: 98.44% | val_loss: 0.9358690204098821 | val_accuracy: 76.62%
[14, 110] loss: 0.0509964578819671155 | accuracy: 98.12% | val_loss: 0.9907288872636855 | val_accuracy: 77.22%
[14, 120] loss: 0.11003897897899151 | accuracy: 96.56% | val_loss: 0.8850810807198286 | val_accuracy: 77.02%
[14, 130] loss: 0.0825831675902009 | accuracy: 96.88% | val_loss: 0.8891354277729988 | val_accuracy: 75.62%
[14, 140] loss: 0.0966179920360446 | accuracy: 97.19% | val_loss: 1.086267976090312 | val_accuracy: 75.42%
[14, 150] loss: 0.17631169483065606 | accuracy: 94.06% | val_loss: 1.3416821043938398 | val_accuracy: 68.53%
[14, 160] loss: 0.32007497027516363 | accuracy: 90.00% | val_loss: 1.2209116648882627 | val_accuracy: 72.13%
[14, 170] loss: 0.34369288086891175 | accuracy: 88.44% | val_loss: 1.230240099132061 | val_accuracy: 69.03%
[14, 180] loss: 0.38305840343236924 | accuracy: 88.44% | val_loss: 1.3252412304282188 | val_accuracy: 67.43%
[15, 10] loss: 0.49573964923620223 | accuracy: 85.94% | val_loss: 1.1905236234888434 | val_accuracy: 69.13%
[15, 20] loss: 0.5185202077031136 | accuracy: 81.56% | val_loss: 1.1884257979691029 | val_accuracy: 67.03%
[15, 30] loss: 0.45216861218214033 | accuracy: 85.62% | val_loss: 1.0893761916086078 | val_accuracy: 68.53%
[15, 40] loss: 0.4720574736595154 | accuracy: 85.31% | val_loss: 0.9197953986003995 | val_accuracy: 72.23%
[15, 50] loss: 0.41242883801460267 | accuracy: 83.75% | val_loss: 1.0666353879496455 | val_accuracy: 71.93%
[15, 60] loss: 0.395389649271965 | accuracy: 83.44% | val_loss: 0.976802496239543 | val_accuracy: 72.63%
[15, 70] loss: 0.2921566590666771 | accuracy: 90.31% | val_loss: 0.8956059133633971 | val_accuracy: 73.73%
[15, 80] loss: 0.31494794860482217 | accuracy: 85.94% | val_loss: 0.8800910674035549 | val_accuracy: 75.52%
[15, 90] loss: 0.33348817825317384 | accuracy: 87.81% | val_loss: 0.8789001628756523 | val_accuracy: 73.73%
[15, 100] loss: 0.2743725150823593 | accuracy: 88.75% | val_loss: 0.9440681338310242 | val_accuracy: 74.53%
[15, 110] loss: 0.2581175535917282 | accuracy: 92.81% | val_loss: 0.8643502136692405 | val_accuracy: 74.03%
[15, 120] loss: 0.24159217476844788 | accuracy: 91.88% | val_loss: 0.883890375494957 | val_accuracy: 74.43%
```

```
[15, 130] loss: 0.21111626625061036 | accuracy: 92.19% | val_loss: 0.8452020939439535 | val_accuracy: 75.62%
[15, 140] loss: 0.20894777700304984 | accuracy: 91.25% | val_loss: 0.9492220850661397 | val_accuracy: 73.63%
[15, 150] loss: 0.21466444283723832 | accuracy: 92.19% | val_loss: 0.8630329100415111 | val_accuracy: 76.32%
[15, 160] loss: 0.1834508553147316 | accuracy: 93.12% | val_loss: 0.9224801063537598 | val_accuracy: 74.93%
[15, 170] loss: 0.2106838312000036 | accuracy: 93.12% | val_loss: 0.859103687107563 | val_accuracy: 75.12%
[15, 180] loss: 0.24233685657382012 | accuracy: 90.94% | val_loss: 0.9086406696587801 | val_accuracy: 74.23%
[16, 10] loss: 0.08733085431158542 | accuracy: 98.12% | val_loss: 0.9539161436259747 | val_accuracy: 75.12%
[16, 20] loss: 0.06425436902791262 | accuracy: 98.75% | val_loss: 0.855805394705385 | val_accuracy: 74.93%
[16, 30] loss: 0.09046242497861386 | accuracy: 96.56% | val_loss: 0.9085665065795183 | val_accuracy: 75.92%
[16, 40] loss: 0.08927472587674856 | accuracy: 96.56% | val_loss: 0.9482476953417063 | val_accuracy: 73.93%
[16, 50] loss: 0.09410128332674503 | accuracy: 97.19% | val_loss: 0.9348852597177029 | val_accuracy: 75.22%
[16, 60] loss: 0.07975856978446245 | accuracy: 97.50% | val_loss: 0.8442891910672188 | val_accuracy: 76.72%
[16, 70] loss: 0.10170113686472178 | accuracy: 96.56% | val_loss: 0.8486360562965274 | val_accuracy: 77.62%
[16, 80] loss: 0.07817066200077534 | accuracy: 97.81% | val_loss: 0.8715338464826345 | val_accuracy: 77.12%
[16, 90] loss: 0.06954825576394796 | accuracy: 98.44% | val_loss: 0.873411831445992 | val_accuracy: 77.32%
[16, 100] loss: 0.0755876999348402 | accuracy: 97.81% | val_loss: 0.9274545875377953 | val_accuracy: 76.42%
[16, 110] loss: 0.0729253027588129 | accuracy: 97.81% | val_loss: 0.8743837997317314 | val_accuracy: 76.02%
[16, 120] loss: 0.12511163540184497 | accuracy: 96.56% | val_loss: 0.9129260182380676 | val_accuracy: 75.62%
[16, 130] loss: 0.07610094640403986 | accuracy: 97.81% | val_loss: 1.0298645375296474 | val_accuracy: 76.72%
[16, 140] loss: 0.07126870704814792 | accuracy: 98.12% | val_loss: 1.0183548610657454 | val_accuracy: 76.92%
[16, 150] loss: 0.07399176768958568 | accuracy: 98.12% | val_loss: 0.8689677864313126 | val_accuracy: 76.42%
[16, 160] loss: 0.06106665097177029 | accuracy: 97.50% | val_loss: 0.9331467002630234 | val_accuracy: 76.02%
[16, 170] loss: 0.05677299704402685 | accuracy: 98.12% | val_loss: 1.0377621054649353 | val_accuracy: 75.42%
[16, 180] loss: 0.0910740016028285 | accuracy: 96.25% | val_loss: 0.9280209178104997 | val_accuracy: 75.52%
[17, 10] loss: 0.041447667963802816 | accuracy: 99.06% | val_loss: 1.1642630454152822 | val_accuracy: 76.52%
[17, 20] loss: 0.026961497962474823 | accuracy: 100.00% | val_loss: 0.9391981400549412 | val_accuracy: 78.02%
[17, 30] loss: 0.04565362446010113 | accuracy: 98.44% | val_loss: 0.9468518458306789 | val_accuracy: 76.82%
[17, 40] loss: 0.02720305062830448 | accuracy: 99.69% | val_loss: 1.060548109933734 | val_accuracy: 75.22%
[17, 50] loss: 0.025791312195360662 | accuracy: 99.69% | val_loss: 0.9041395271196961 | val_accuracy: 76.62%
[17, 60] loss: 0.030939908465370536 | accuracy: 98.75% | val_loss: 0.8954761540517211 | val_accuracy: 76.52%
[17, 70] loss: 0.025526810600422323 | accuracy: 98.75% | val_loss: 1.0027928375639021 | val_accuracy: 77.52%
[17, 80] loss: 0.022513466235250235 | accuracy: 99.69% | val_loss: 0.9466279679909348 | val_accuracy: 77.52%
[17, 90] loss: 0.025520556280389427 | accuracy: 99.38% | val_loss: 1.0115755246952176 | val_accuracy: 76.82%
[17, 100] loss: 0.026385349314659835 | accuracy: 99.69% | val_loss: 1.0579945258796215 | val_accuracy: 76.82%
[17, 110] loss: 0.028022194560617721 | accuracy: 99.06% | val_loss: 1.0378834512084723 | val_accuracy: 76.52%
[17, 120] loss: 0.03155543589964509 | accuracy: 99.38% | val_loss: 0.9523917756741866 | val_accuracy: 76.22%
[17, 130] loss: 0.029002569895237685 | accuracy: 99.69% | val_loss: 1.0290159238502383 | val_accuracy: 76.92%
[17, 140] loss: 0.02098571276292205 | accuracy: 100.00% | val_loss: 1.0427995431236923 | val_accuracy: 77.82%
[17, 150] loss: 0.03567408134695142 | accuracy: 98.75% | val_loss: 1.0344724897295237 | val_accuracy: 76.12%
[17, 160] loss: 0.023623794317245483 | accuracy: 99.69% | val_loss: 1.096784452907741 | val_accuracy: 74.63%
[17, 170] loss: 0.027889063768088817 | accuracy: 99.06% | val_loss: 1.0636102235876024 | val_accuracy: 76.12%
[17, 180] loss: 0.03421892351470888 | accuracy: 99.69% | val_loss: 1.0321167488582432 | val_accuracy: 75.82%
```

```
[18, 10] loss: 0.04400113318115473 | accuracy: 97.81% | val_loss: 1.1583909673208836 | val_accuracy: 74.93%
[18, 20] loss: 0.042383333574980496 | accuracy: 98.44% | val_loss: 1.1642248288262635 | val_accuracy: 73.93%
[18, 30] loss: 0.021744605619460344 | accuracy: 100.00% | val_loss: 1.09319750021958351 | val_accuracy: 74.93%
[18, 40] loss: 0.039773119427263734 | accuracy: 99.06% | val_loss: 1.047281626611948 | val_accuracy: 75.02%
[18, 50] loss: 0.03474041782319546 | accuracy: 98.44% | val_loss: 1.106070772279054 | val_accuracy: 76.42%
[18, 60] loss: 0.03608318879269064 | accuracy: 98.75% | val_loss: 1.1108182622119784 | val_accuracy: 77.22%
[18, 70] loss: 0.045808013435453175 | accuracy: 97.81% | val_loss: 1.1047103116288781 | val_accuracy: 75.82%
[18, 80] loss: 0.05174839673563838 | accuracy: 98.75% | val_loss: 1.0082464395090938 | val_accuracy: 76.52%
[18, 90] loss: 0.05389313846826553 | accuracy: 98.44% | val_loss: 1.0428943894803524 | val_accuracy: 76.12%
[18, 100] loss: 0.03801048304885626 | accuracy: 98.44% | val_loss: 1.1771902267355472 | val_accuracy: 77.32%
[18, 110] loss: 0.06311619449406862 | accuracy: 98.12% | val_loss: 1.1226431066170335 | val_accuracy: 75.72%
[18, 120] loss: 0.07573584276251495 | accuracy: 97.50% | val_loss: 0.9691665912978351 | val_accuracy: 77.32%
[18, 130] loss: 0.041198386205360295 | accuracy: 98.75% | val_loss: 1.0499125043861568 | val_accuracy: 77.22%
[18, 140] loss: 0.044523959048092365 | accuracy: 98.75% | val_loss: 1.1034487765282393 | val_accuracy: 76.02%
[18, 150] loss: 0.028677542693912983 | accuracy: 99.06% | val_loss: 1.0832493295893073 | val_accuracy: 75.42%
[18, 160] loss: 0.033233076147735116 | accuracy: 98.75% | val_loss: 1.1513370610773563 | val_accuracy: 76.32%
[18, 170] loss: 0.05017867516726256 | accuracy: 98.12% | val_loss: 1.156607878394425 | val_accuracy: 76.02%
[18, 180] loss: 0.03585084555670619 | accuracy: 99.38% | val_loss: 1.153944669291377 | val_accuracy: 76.22%
[19, 10] loss: 0.03282549101859331 | accuracy: 98.44% | val_loss: 1.0780758364126086 | val_accuracy: 76.02%
[19, 20] loss: 0.024516710732132196 | accuracy: 99.38% | val_loss: 1.1376294349320233 | val_accuracy: 76.52%
[19, 30] loss: 0.024862885614857076 | accuracy: 99.38% | val_loss: 1.1609821543097496 | val_accuracy: 76.92%
[19, 40] loss: 0.035346540971659124 | accuracy: 98.44% | val_loss: 1.1922883912920952 | val_accuracy: 77.02%
[19, 50] loss: 0.031171096954494715 | accuracy: 99.69% | val_loss: 1.2085969243198633 | val_accuracy: 76.52%
[19, 60] loss: 0.04109780993312597 | accuracy: 98.75% | val_loss: 1.297060564160347 | val_accuracy: 75.52%
[19, 70] loss: 0.05811179550364613 | accuracy: 98.12% | val_loss: 1.210824977606535 | val_accuracy: 76.22%
[19, 80] loss: 0.036411972250789404 | accuracy: 98.75% | val_loss: 1.0613366484176368 | val_accuracy: 75.82%
[19, 90] loss: 0.03166766818612814 | accuracy: 99.06% | val_loss: 1.159876998513937 | val_accuracy: 75.52%
[19, 100] loss: 0.04073687865165994 | accuracy: 98.12% | val_loss: 1.1419364856556058 | val_accuracy: 75.32%
[19, 110] loss: 0.037305661290884015 | accuracy: 98.75% | val_loss: 1.0927963587455451 | val_accuracy: 76.42%
[19, 120] loss: 0.046633616032078862 | accuracy: 98.75% | val_loss: 1.3095095697790384 | val_accuracy: 76.42%
[19, 130] loss: 0.0330146221909672 | accuracy: 99.06% | val_loss: 1.3624939247965813 | val_accuracy: 76.52%
[19, 140] loss: 0.04918899382464588 | accuracy: 98.44% | val_loss: 1.263733577914536 | val_accuracy: 75.92%
[19, 150] loss: 0.01980252468492836 | accuracy: 99.69% | val_loss: 1.2078442415222526 | val_accuracy: 75.22%
[19, 160] loss: 0.04361733673140407 | accuracy: 98.12% | val_loss: 1.1092464574612677 | val_accuracy: 74.33%
[19, 170] loss: 0.028691383730620145 | accuracy: 99.38% | val_loss: 1.2624807432293892 | val_accuracy: 75.92%
[19, 180] loss: 0.03292649185750633 | accuracy: 98.44% | val_loss: 1.590342273004353 | val_accuracy: 75.42%
[20, 10] loss: 0.02175521143944934 | accuracy: 99.38% | val_loss: 1.2991049876436591 | val_accuracy: 75.82%
[20, 20] loss: 0.031927933520637455 | accuracy: 99.38% | val_loss: 1.1845022700726986 | val_accuracy: 75.92%
[20, 30] loss: 0.0410740529187023064 | accuracy: 98.12% | val_loss: 1.232221613638103 | val_accuracy: 74.13%
[20, 40] loss: 0.0348235527984798 | accuracy: 98.75% | val_loss: 1.2137351660057902 | val_accuracy: 75.32%
[20, 50] loss: 0.04165551303885877 | accuracy: 99.06% | val_loss: 1.2063562273979187 | val_accuracy: 73.93%
[20, 60] loss: 0.05446967093739659 | accuracy: 98.12% | val_loss: 1.411851180717349 | val_accuracy: 74.93%
```

```
[20, 70] loss: 0.06584543150383979 | accuracy: 98.75% | val_loss: 1.2757055163383484 | val_accuracy: 74.13%
[20, 80] loss: 0.04780685137957334 | accuracy: 98.12% | val_loss: 1.2773096915334463 | val_accuracy: 73.43%
[20, 90] loss: 0.02913059606216848 | accuracy: 99.06% | val_loss: 1.2940648831427097 | val_accuracy: 74.83%
[20, 100] loss: 0.03795592403039336 | accuracy: 98.75% | val_loss: 1.4006359623817843 | val_accuracy: 75.62%
[20, 110] loss: 0.06953518060036004 | accuracy: 97.81% | val_loss: 1.2208834467455745 | val_accuracy: 77.42%
[20, 120] loss: 0.07346122339367867 | accuracy: 97.81% | val_loss: 1.1770339645445347 | val_accuracy: 76.32%
[20, 130] loss: 0.03822103044949472 | accuracy: 99.06% | val_loss: 1.242755758576095 | val_accuracy: 75.82%
[20, 140] loss: 0.03869967209175229 | accuracy: 98.75% | val_loss: 1.226339160464704 | val_accuracy: 74.23%
[20, 150] loss: 0.039054635399952534 | accuracy: 98.44% | val_loss: 1.2944226926192641 | val_accuracy: 74.83%
[20, 160] loss: 0.05315618868917227 | accuracy: 98.12% | val_loss: 1.2595800887793303 | val_accuracy: 75.72%
[20, 170] loss: 0.045203320204745975 | accuracy: 98.12% | val_loss: 1.4370029252022505 | val_accuracy: 75.22%
[20, 180] loss: 0.03638701136223972 | accuracy: 98.44% | val_loss: 1.2892804685980082 | val_accuracy: 75.82%
```

```python
In [34]:  # Get the number_to_label_array so that we can use it later to plot the confusion matrix
          complex_number_to_label_array = complex_testloader.dataset.dataset.number_to_label_array
          # Test the model
          test_net(complex_model, complex_testloader,complex_number_to_label_array, loss=complex_loss, device=device)
```

```
Test loss: 1.1766993999481201 | Test accuracy: 77.00%
```

Confusion Matrix
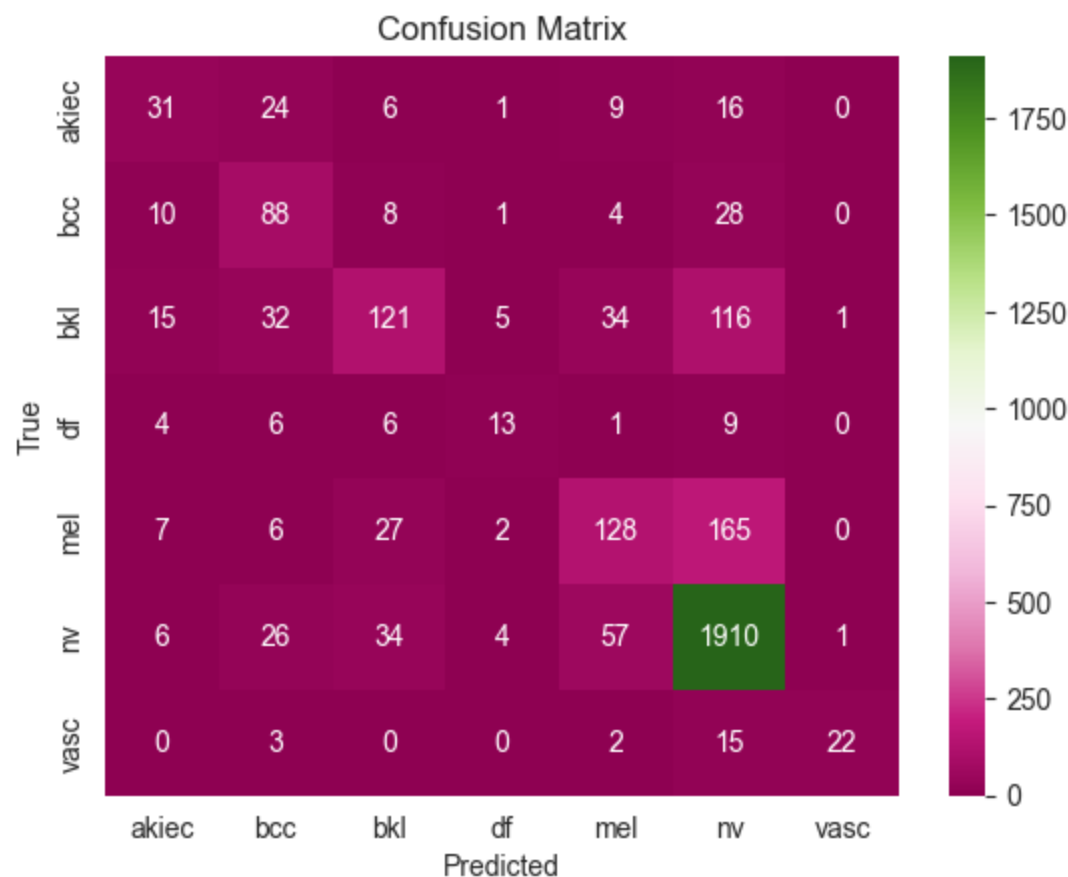
Observations

- We can see again that the model has a strong inclination to predict the label `nv` , but this time not so much for `mel`
- The model predictions are mostly concentrated on the diagonal, (correct predictions), which is a VERY good thing
- There is also a huge improvement on `df` , the model actually predicted the label , and it did it correctly most of the time.
- Interestingly, for the `vasc` label, the model is not as good as the Simple model, precision is still good, but the recall is not as good as the Simple model

```
In [18]:  # Copying the lists so that I don't overwrite them
          complex_train_loss = copy.deepcopy(train_loss)
          complex_val_loss = copy.deepcopy(val_loss)
```

```
complex_train_accuracy = copy.deepcopy(train_accuracy)
complex_val_accuracy = copy.deepcopy(val_accuracy)
```

In [19]:
```
x_axis = list(map(lambda x:x*10,range(len(complex_train_loss))))
plt.figure(figsize=(10,6))
plt.plot(x_axis ,complex_train_loss, label='Training Loss')
plt.plot(x_axis , complex_val_loss,label='Validation Loss')
xticks = np.arange(x_axis[0], x_axis[-1]+1, 400.0)
plt.xticks([*xticks, x_axis[-1]])
plt.xlabel('Batch Number')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.grid()
plt.show()
```
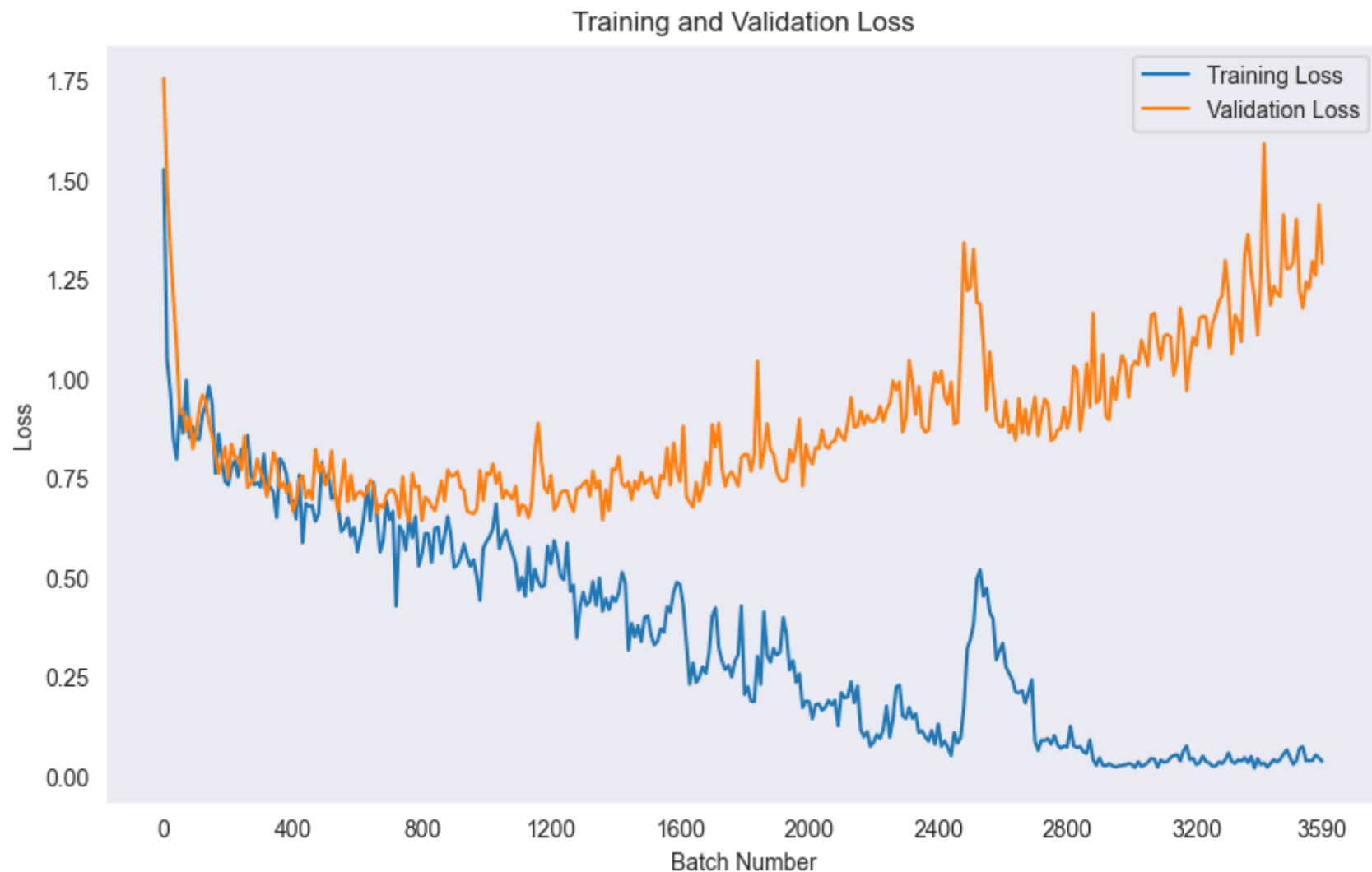
Observations

- We can see that the model starts to overfit at around 400-500 batches, as the validation loss starts to increase, while the training loss keeps decreasing
- We can see that the graph is less "jagged", which I think means that the model isn't overshooting the minimum loss, meaning the optimizer setting and choice are way better than the ones for the Simple model.

- There is a weird spike at 2400 batches, which I think might mean that the while the optimizer was steadily decreasing the loss, there was a sudden "hill" on the feature space.
- This graph indicates that the model is doing two things better than the Simple model, it is not overshooting the minimum loss (the optimizer choice and settings are way better), and the model starts overfitting faster, which means that the model is learning faster.

In [20]:
```python
x_axis = list(map(lambda x:x*10,range(len(complex_train_accuracy))))
plt.figure(figsize=(10,6))
plt.plot(x_axis ,complex_train_accuracy, label='Training Accuracy')
plt.plot(x_axis , complex_val_accuracy,label='Validation Accuracy')
xticks = np.arange(x_axis[0], x_axis[-1]+1, 200.0)
plt.xticks([*xticks, x_axis[-1]])
plt.xlabel('Batch Number')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.grid()
plt.show()
```

Training and Validation Accuracy
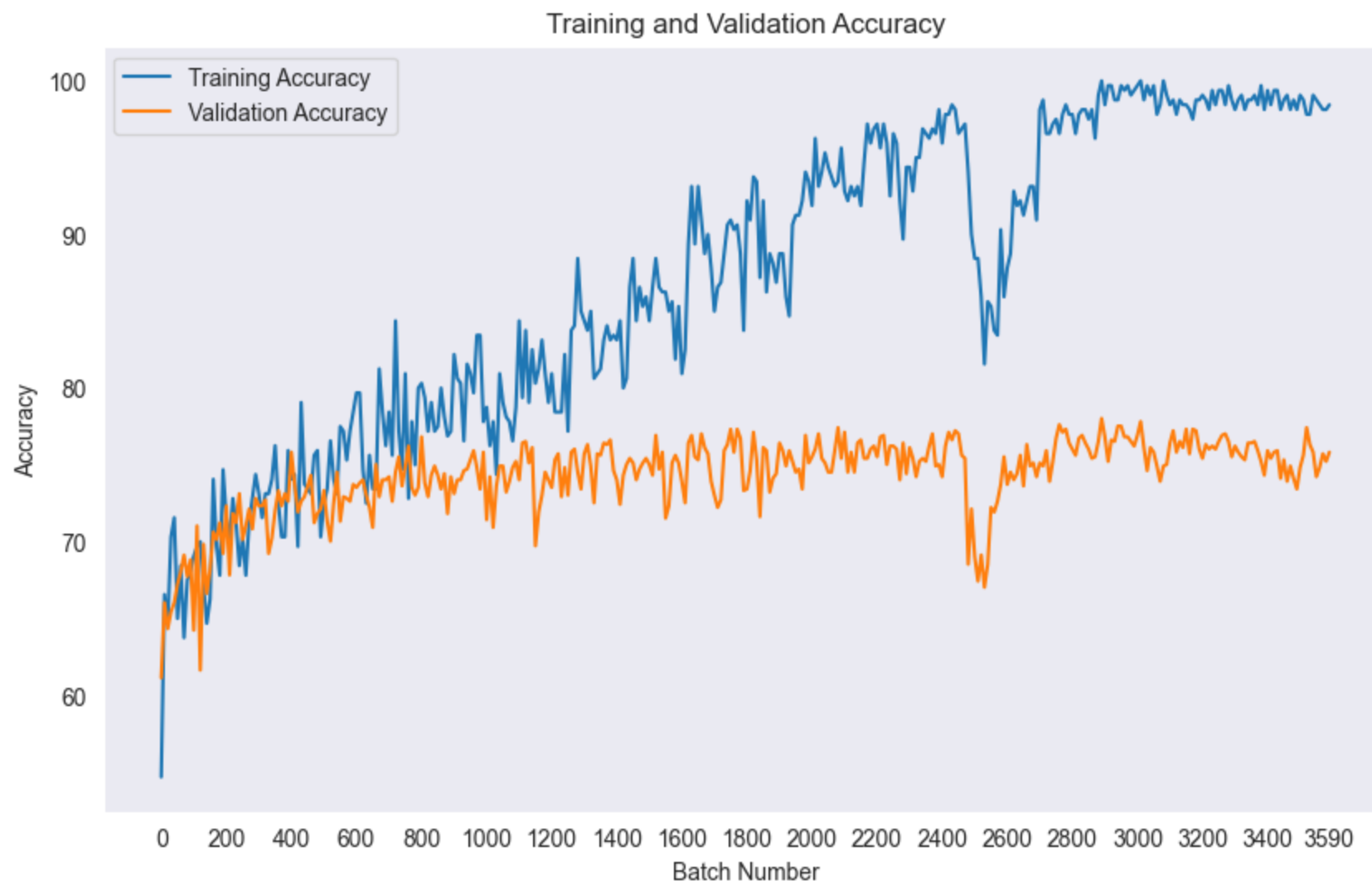
Observations

- In this graph the overfitting is really visible, because the validation accuracy is not decreasing, but it is not increasing either, while the training accuracy is increasing.
- This time it also less "jagged"
- and we can still see the same weird spike at 2400 batches

```python
In [21]:  # I will also save the models so that I can load them later instead of training them again (takes a lot of time)
          torch.save(model.state_dict(), 'results/model_states/simple_model.pth')
          torch.save(complex_model.state_dict(), 'results/model_states/complex_model.pth')

          # We don't need to save the dataloaders, because we use the same seed every time
          # so the train, validation and test sets will be the same (seed = 42)
          # But we do need to save the train, validation loss and accuracy
          # so that we can plot them later without having to train the models again

          torch.save(simple_train_loss, 'results/general/simple_train_loss.list.pyobj')
          torch.save(simple_val_loss, 'results/general/simple_val_loss.list.pyobj')
          torch.save(simple_train_accuracy, 'results/general/simple_train_accuracy.list.pyobj')
          torch.save(simple_val_accuracy, 'results/general/simple_val_accuracy.list.pyobj')

          torch.save(complex_train_loss, 'results/general/complex_train_loss.list.pyobj')
          torch.save(complex_val_loss, 'results/general/complex_val_loss.list.pyobj')
          torch.save(complex_train_accuracy, 'results/general/complex_train_accuracy.list.pyobj')
          torch.save(complex_val_accuracy, 'results/general/complex_val_accuracy.list.pyobj')
```

ResNet: Transfer Learning

```python
In [22]:  import torchvision.transforms as transforms
          # The transforms for the ResNet model are different from the ones for the Simple and Complex models
          # create the transforms as given in the assignment
          data_transforms = {
                  'train': torchvision.transforms.Compose([
                      transforms.RandomResizedCrop(224),
                      transforms.RandomHorizontalFlip(),
                      transforms.ToTensor(),
                      transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225] )]),
                  'val': torchvision.transforms.Compose([
                      transforms.Resize(256),
                      transforms.CenterCrop(224),
                      transforms.ToTensor(),
                      transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225] )])
              ,
              }

          # Create the dataset
```

```python
resnet_dataset = MLProject2Dataset(data_dir='data', metadata_fname='metadata.csv')

# To apply the different transforms to the train, validation and test sets
# We need to first split the dataset into train, validation and test to separate them as python objects
# and then apply the transforms to each one of them
# The split is train:val:test = 60%:10%:30% and because we use different transform for the train
# and validation/test sets, we need to split into train:val+test = 60%:40% and then split the val+test
resnet_train_dataset, resnet_val_test_dataset = torch.utils.data.random_split(resnet_dataset, [.6, .4],
                                                            generator=torch.Generator().manual_seed(42)
                                                            )

# Set the transforms for the train set
resnet_train_dataset.dataset.transform = data_transforms['train']
# Split the val+test set into val and test
# the split in the original dataset is 10%:30% when the total of the two is 40%
# now that the "total" of the two is 100%, the split is 25%:75% to get the correct split
resnet_val_dataset, resnet_test_dataset = torch.utils.data.random_split(resnet_val_test_dataset, [.25, .75],
                                                            generator=torch.Generator().manual_seed(42)
                                                            )

# Set the transforms for the val and test sets
# I don't really need to do the transform setting for both of them, because they are the same python object
# under the hood, but I do it for readability, and just in case that the implementation of the random_split
# changes in the future
resnet_val_dataset.dataset.transform = data_transforms['val']
resnet_test_dataset.dataset.transform = data_transforms['val']

# Create the dataloaders
BATCH_SIZE = 32
resnet_trainloader = DataLoader(resnet_train_dataset, batch_size=BATCH_SIZE, shuffle=True)
resnet_valloader = DataLoader(resnet_val_dataset, batch_size=BATCH_SIZE, shuffle=True)
resnet_testloader = DataLoader(resnet_test_dataset, batch_size=BATCH_SIZE, shuffle=True)

# Get the pretrained model
resnet_model = torchvision.models.resnet34(weights="DEFAULT")
resnet_model = resnet_model.to(device)
# Print the model summary
summary(resnet_model, (3, 224, 224))
# Create the optimizer
resnet_optimizer = optim.SGD(resnet_model.parameters(), lr=0.001,momentum=0.9)
# Create the loss function
resnet_loss = nn.CrossEntropyLoss()
# Train the model
```

```python
train_net(resnet_model, resnet_trainloader,resnet_valloader,epochs=5, optimizer=resnet_optimizer,
          loss=resnet_loss, device=device)
# This cell took 17 minutes to run on my machine
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
          Conv2d-1           [-1, 64, 112, 112]           9,408
     BatchNorm2d-2           [-1, 64, 112, 112]             128
            ReLU-3           [-1, 64, 112, 112]               0
       MaxPool2d-4             [-1, 64, 56, 56]               0
          Conv2d-5             [-1, 64, 56, 56]          36,864
     BatchNorm2d-6             [-1, 64, 56, 56]             128
            ReLU-7             [-1, 64, 56, 56]               0
          Conv2d-8             [-1, 64, 56, 56]          36,864
     BatchNorm2d-9             [-1, 64, 56, 56]             128
           ReLU-10             [-1, 64, 56, 56]               0
     BasicBlock-11             [-1, 64, 56, 56]               0
         Conv2d-12             [-1, 64, 56, 56]          36,864
    BatchNorm2d-13             [-1, 64, 56, 56]             128
           ReLU-14             [-1, 64, 56, 56]               0
         Conv2d-15             [-1, 64, 56, 56]          36,864
    BatchNorm2d-16             [-1, 64, 56, 56]             128
           ReLU-17             [-1, 64, 56, 56]               0
     BasicBlock-18             [-1, 64, 56, 56]               0
         Conv2d-19             [-1, 64, 56, 56]          36,864
    BatchNorm2d-20             [-1, 64, 56, 56]             128
           ReLU-21             [-1, 64, 56, 56]               0
         Conv2d-22             [-1, 64, 56, 56]          36,864
    BatchNorm2d-23             [-1, 64, 56, 56]             128
           ReLU-24             [-1, 64, 56, 56]               0
     BasicBlock-25             [-1, 64, 56, 56]               0
         Conv2d-26            [-1, 128, 28, 28]          73,728
    BatchNorm2d-27            [-1, 128, 28, 28]             256
           ReLU-28            [-1, 128, 28, 28]               0
         Conv2d-29            [-1, 128, 28, 28]         147,456
    BatchNorm2d-30            [-1, 128, 28, 28]             256
         Conv2d-31            [-1, 128, 28, 28]           8,192
    BatchNorm2d-32            [-1, 128, 28, 28]             256
           ReLU-33            [-1, 128, 28, 28]               0
     BasicBlock-34            [-1, 128, 28, 28]               0
         Conv2d-35            [-1, 128, 28, 28]         147,456
    BatchNorm2d-36            [-1, 128, 28, 28]             256
           ReLU-37            [-1, 128, 28, 28]               0
         Conv2d-38            [-1, 128, 28, 28]         147,456
    BatchNorm2d-39            [-1, 128, 28, 28]             256
```

```
           ReLU-40          [-1, 128, 28, 28]               0
    BasicBlock-41          [-1, 128, 28, 28]               0
        Conv2d-42          [-1, 128, 28, 28]         147,456
   BatchNorm2d-43          [-1, 128, 28, 28]             256
           ReLU-44          [-1, 128, 28, 28]               0
        Conv2d-45          [-1, 128, 28, 28]         147,456
   BatchNorm2d-46          [-1, 128, 28, 28]             256
           ReLU-47          [-1, 128, 28, 28]               0
    BasicBlock-48          [-1, 128, 28, 28]               0
        Conv2d-49          [-1, 128, 28, 28]         147,456
   BatchNorm2d-50          [-1, 128, 28, 28]             256
           ReLU-51          [-1, 128, 28, 28]               0
        Conv2d-52          [-1, 128, 28, 28]         147,456
   BatchNorm2d-53          [-1, 128, 28, 28]             256
           ReLU-54          [-1, 128, 28, 28]               0
    BasicBlock-55          [-1, 128, 28, 28]               0
        Conv2d-56          [-1, 256, 14, 14]         294,912
   BatchNorm2d-57          [-1, 256, 14, 14]             512
           ReLU-58          [-1, 256, 14, 14]               0
        Conv2d-59          [-1, 256, 14, 14]         589,824
   BatchNorm2d-60          [-1, 256, 14, 14]             512
        Conv2d-61          [-1, 256, 14, 14]          32,768
   BatchNorm2d-62          [-1, 256, 14, 14]             512
           ReLU-63          [-1, 256, 14, 14]               0
    BasicBlock-64          [-1, 256, 14, 14]               0
        Conv2d-65          [-1, 256, 14, 14]         589,824
   BatchNorm2d-66          [-1, 256, 14, 14]             512
           ReLU-67          [-1, 256, 14, 14]               0
        Conv2d-68          [-1, 256, 14, 14]         589,824
   BatchNorm2d-69          [-1, 256, 14, 14]             512
           ReLU-70          [-1, 256, 14, 14]               0
    BasicBlock-71          [-1, 256, 14, 14]               0
        Conv2d-72          [-1, 256, 14, 14]         589,824
   BatchNorm2d-73          [-1, 256, 14, 14]             512
           ReLU-74          [-1, 256, 14, 14]               0
        Conv2d-75          [-1, 256, 14, 14]         589,824
   BatchNorm2d-76          [-1, 256, 14, 14]             512
           ReLU-77          [-1, 256, 14, 14]               0
    BasicBlock-78          [-1, 256, 14, 14]               0
        Conv2d-79          [-1, 256, 14, 14]         589,824
   BatchNorm2d-80          [-1, 256, 14, 14]             512
           ReLU-81          [-1, 256, 14, 14]               0
```

```
          Conv2d-82           [-1, 256, 14, 14]          589,824
     BatchNorm2d-83           [-1, 256, 14, 14]              512
            ReLU-84           [-1, 256, 14, 14]                0
      BasicBlock-85           [-1, 256, 14, 14]                0
          Conv2d-86           [-1, 256, 14, 14]          589,824
     BatchNorm2d-87           [-1, 256, 14, 14]              512
            ReLU-88           [-1, 256, 14, 14]                0
          Conv2d-89           [-1, 256, 14, 14]          589,824
     BatchNorm2d-90           [-1, 256, 14, 14]              512
            ReLU-91           [-1, 256, 14, 14]                0
      BasicBlock-92           [-1, 256, 14, 14]                0
          Conv2d-93           [-1, 256, 14, 14]          589,824
     BatchNorm2d-94           [-1, 256, 14, 14]              512
            ReLU-95           [-1, 256, 14, 14]                0
          Conv2d-96           [-1, 256, 14, 14]          589,824
     BatchNorm2d-97           [-1, 256, 14, 14]              512
            ReLU-98           [-1, 256, 14, 14]                0
      BasicBlock-99           [-1, 256, 14, 14]                0
         Conv2d-100            [-1, 512, 7, 7]        1,179,648
    BatchNorm2d-101            [-1, 512, 7, 7]            1,024
           ReLU-102            [-1, 512, 7, 7]                0
         Conv2d-103            [-1, 512, 7, 7]        2,359,296
    BatchNorm2d-104            [-1, 512, 7, 7]            1,024
         Conv2d-105            [-1, 512, 7, 7]          131,072
    BatchNorm2d-106            [-1, 512, 7, 7]            1,024
           ReLU-107            [-1, 512, 7, 7]                0
     BasicBlock-108            [-1, 512, 7, 7]                0
         Conv2d-109            [-1, 512, 7, 7]        2,359,296
    BatchNorm2d-110            [-1, 512, 7, 7]            1,024
           ReLU-111            [-1, 512, 7, 7]                0
         Conv2d-112            [-1, 512, 7, 7]        2,359,296
    BatchNorm2d-113            [-1, 512, 7, 7]            1,024
           ReLU-114            [-1, 512, 7, 7]                0
     BasicBlock-115            [-1, 512, 7, 7]                0
         Conv2d-116            [-1, 512, 7, 7]        2,359,296
    BatchNorm2d-117            [-1, 512, 7, 7]            1,024
           ReLU-118            [-1, 512, 7, 7]                0
         Conv2d-119            [-1, 512, 7, 7]        2,359,296
    BatchNorm2d-120            [-1, 512, 7, 7]            1,024
           ReLU-121            [-1, 512, 7, 7]                0
     BasicBlock-122            [-1, 512, 7, 7]                0
AdaptiveAvgPool2d-123           [-1, 512, 1, 1]                0
```

```
            Linear-124              [-1, 1000]          513,000
================================================================
Total params: 21,797,672
Trainable params: 21,797,672
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 96.29
Params size (MB): 83.15
Estimated Total Size (MB): 180.01
----------------------------------------------------------------
[1, 10] loss: 6.17704451084137 | accuracy: 17.19% | val_loss: 3.963384687900543 | val_accuracy: 58.38%
[1, 20] loss: 2.2898713290691375 | accuracy: 67.19% | val_loss: 2.5974833779037 | val_accuracy: 66.07%
[1, 30] loss: 1.4942876398563385 | accuracy: 63.12% | val_loss: 1.596372740343213 | val_accuracy: 65.97%
[1, 40] loss: 1.6530644059181214 | accuracy: 63.12% | val_loss: 1.1404171604663134 | val_accuracy: 70.56%
[1, 50] loss: 1.1083948910236359 | accuracy: 67.50% | val_loss: 0.9946752395480871 | val_accuracy: 69.66%
[1, 60] loss: 0.9289926886558533 | accuracy: 69.06% | val_loss: 0.8834453672170639 | val_accuracy: 71.16%
[1, 70] loss: 0.9626702964305878 | accuracy: 69.38% | val_loss: 0.8547184336930513 | val_accuracy: 73.25%
[1, 80] loss: 0.9119177520275116 | accuracy: 70.94% | val_loss: 0.8553326427936554 | val_accuracy: 73.05%
[1, 90] loss: 0.7668964982032775 | accuracy: 71.56% | val_loss: 0.8221858739852905 | val_accuracy: 73.55%
[1, 100] loss: 0.7962729752063751 | accuracy: 72.19% | val_loss: 0.8114949213340878 | val_accuracy: 73.55%
[1, 110] loss: 0.9719425022602082 | accuracy: 71.25% | val_loss: 0.7864045165479183 | val_accuracy: 73.15%
[1, 120] loss: 0.736609798669815 | accuracy: 74.38% | val_loss: 0.7609227607026696 | val_accuracy: 73.65%
[1, 130] loss: 0.7409658372402191 | accuracy: 70.62% | val_loss: 0.7462992370128632 | val_accuracy: 73.45%
[1, 140] loss: 0.8091832995414734 | accuracy: 73.75% | val_loss: 0.7429159078747034 | val_accuracy: 73.45%
[1, 150] loss: 0.7454368084669113 | accuracy: 73.75% | val_loss: 0.7233982002362609 | val_accuracy: 75.05%
[1, 160] loss: 0.6959013700485229 | accuracy: 74.38% | val_loss: 0.7358564985916018 | val_accuracy: 73.85%
[1, 170] loss: 0.8508499145507813 | accuracy: 69.69% | val_loss: 0.7413502847775817 | val_accuracy: 73.55%
[1, 180] loss: 0.846804392337799 | accuracy: 69.06% | val_loss: 0.7032450865954161 | val_accuracy: 75.75%
[2, 10] loss: 0.6396559566259384 | accuracy: 80.00% | val_loss: 0.6940733138471842 | val_accuracy: 76.25%
[2, 20] loss: 0.6204664379358291 | accuracy: 79.38% | val_loss: 0.7163570560514927 | val_accuracy: 74.55%
[2, 30] loss: 0.6742629557847977 | accuracy: 73.44% | val_loss: 0.7164107616990805 | val_accuracy: 75.05%
[2, 40] loss: 0.7504343122243882 | accuracy: 70.31% | val_loss: 0.6717722648754716 | val_accuracy: 76.45%
[2, 50] loss: 0.655427747964859 | accuracy: 78.44% | val_loss: 0.6752499169670045 | val_accuracy: 75.55%
[2, 60] loss: 0.6402925223112106 | accuracy: 76.25% | val_loss: 0.6796210128813982 | val_accuracy: 76.45%
[2, 70] loss: 0.7026095628738404 | accuracy: 76.56% | val_loss: 0.6796881649643183 | val_accuracy: 73.65%
[2, 80] loss: 0.6857865035533905 | accuracy: 76.25% | val_loss: 0.6736368164420128 | val_accuracy: 75.75%
[2, 90] loss: 0.6046379715204239 | accuracy: 76.25% | val_loss: 0.6507622916251421 | val_accuracy: 76.05%
[2, 100] loss: 0.7441140681505203 | accuracy: 75.00% | val_loss: 0.6206182185560465 | val_accuracy: 77.84%
[2, 110] loss: 0.6700194746255874 | accuracy: 75.00% | val_loss: 0.6401665164157748 | val_accuracy: 76.55%
[2, 120] loss: 0.6947496980428696 | accuracy: 76.88% | val_loss: 0.6129825785756111 | val_accuracy: 77.45%
[2, 130] loss: 0.6946976810693741 | accuracy: 75.00% | val_loss: 0.6587105905637145 | val_accuracy: 75.55%
```

```
[2, 140] loss: 0.7735829949378967 | accuracy: 70.94% | val_loss: 0.6573610911145806 | val_accuracy: 75.85%
[2, 150] loss: 0.6324433147907257 | accuracy: 78.12% | val_loss: 0.6592126293107867 | val_accuracy: 76.45%
[2, 160] loss: 0.6528333395719528 | accuracy: 73.75% | val_loss: 0.6521692797541618 | val_accuracy: 75.95%
[2, 170] loss: 0.6255364865064621 | accuracy: 77.50% | val_loss: 0.6202802825719118 | val_accuracy: 76.55%
[2, 180] loss: 0.6357477635145188 | accuracy: 76.25% | val_loss: 0.6354810670018196 | val_accuracy: 77.54%
[3, 10] loss: 0.6040563881397247 | accuracy: 80.31% | val_loss: 0.6221648240461946 | val_accuracy: 76.75%
[3, 20] loss: 0.5969482988119126 | accuracy: 76.56% | val_loss: 0.6219233246520162 | val_accuracy: 76.95%
[3, 30] loss: 0.6259032398462295 | accuracy: 77.19% | val_loss: 0.5958798145875335 | val_accuracy: 77.84%
[3, 40] loss: 0.6345108032226563 | accuracy: 76.56% | val_loss: 0.6046195938251913 | val_accuracy: 78.24%
[3, 50] loss: 0.6221473395824433 | accuracy: 75.31% | val_loss: 0.5879403268918395 | val_accuracy: 79.64%
[3, 60] loss: 0.613570237159729 | accuracy: 75.94% | val_loss: 0.5908072791062295 | val_accuracy: 80.24%
[3, 70] loss: 0.5462081611156464 | accuracy: 80.62% | val_loss: 0.5837200777605176 | val_accuracy: 78.94%
[3, 80] loss: 0.5016478598117828 | accuracy: 82.19% | val_loss: 0.5675673540681601 | val_accuracy: 79.24%
[3, 90] loss: 0.4605780065059662 | accuracy: 83.75% | val_loss: 0.5951984468847513 | val_accuracy: 78.54%
[3, 100] loss: 0.619431909918785 | accuracy: 78.12% | val_loss: 0.5446687014773488 | val_accuracy: 80.14%
[3, 110] loss: 0.5475737601518631 | accuracy: 79.69% | val_loss: 0.5427374430000782 | val_accuracy: 80.54%
[3, 120] loss: 0.5612233310937882 | accuracy: 79.38% | val_loss: 0.5757134179584682 | val_accuracy: 78.64%
[3, 130] loss: 0.5990054726600647 | accuracy: 78.12% | val_loss: 0.5720548103563488 | val_accuracy: 79.24%
[3, 140] loss: 0.5461213797330856 | accuracy: 81.25% | val_loss: 0.5740677807480097 | val_accuracy: 79.94%
[3, 150] loss: 0.58452088534832 | accuracy: 79.69% | val_loss: 0.6013528611510992 | val_accuracy: 78.94%
[3, 160] loss: 0.6221756607294082 | accuracy: 77.81% | val_loss: 0.6221334505826235 | val_accuracy: 77.15%
[3, 170] loss: 0.5707456797361374 | accuracy: 79.69% | val_loss: 0.6094980393536389 | val_accuracy: 78.54%
[3, 180] loss: 0.6620981216430664 | accuracy: 72.19% | val_loss: 0.5457635661587119 | val_accuracy: 80.84%
[4, 10] loss: 0.5123346477746964 | accuracy: 78.75% | val_loss: 0.57872419571504 | val_accuracy: 80.44%
[4, 20] loss: 0.5973245888948441 | accuracy: 78.44% | val_loss: 0.5627824813127518 | val_accuracy: 79.54%
[4, 30] loss: 0.49759613573551176 | accuracy: 80.00% | val_loss: 0.5711343346629292 | val_accuracy: 78.84%
[4, 40] loss: 0.5610623598098755 | accuracy: 79.38% | val_loss: 0.6476021381095052 | val_accuracy: 77.25%
[4, 50] loss: 0.6056106597185135 | accuracy: 79.06% | val_loss: 0.5813678400591016 | val_accuracy: 78.94%
[4, 60] loss: 0.5589507460594177 | accuracy: 78.12% | val_loss: 0.5889836647547781 | val_accuracy: 79.14%
[4, 70] loss: 0.4521818935871124 | accuracy: 83.75% | val_loss: 0.6049134060740471 | val_accuracy: 78.74%
[4, 80] loss: 0.5941673874855041 | accuracy: 79.06% | val_loss: 0.5618166085332632 | val_accuracy: 79.34%
[4, 90] loss: 0.5632505685091018 | accuracy: 80.94% | val_loss: 0.5733890561386943 | val_accuracy: 78.74%
[4, 100] loss: 0.5857667058706284 | accuracy: 80.00% | val_loss: 0.5667867623269558 | val_accuracy: 78.74%
[4, 110] loss: 0.5679505437612533 | accuracy: 78.12% | val_loss: 0.5536642561201006 | val_accuracy: 79.84%
[4, 120] loss: 0.47972066700458527 | accuracy: 84.06% | val_loss: 0.601851143874228 | val_accuracy: 77.94%
[4, 130] loss: 0.4612834334373474 | accuracy: 82.50% | val_loss: 0.557410279288888 | val_accuracy: 79.84%
[4, 140] loss: 0.6321047872304917 | accuracy: 76.25% | val_loss: 0.5484148599207401 | val_accuracy: 80.44%
[4, 150] loss: 0.5147823840379715 | accuracy: 80.00% | val_loss: 0.565932338591665 | val_accuracy: 78.14%
[4, 160] loss: 0.6062120109796524 | accuracy: 78.12% | val_loss: 0.5562903340905905 | val_accuracy: 79.44%
[4, 170] loss: 0.6271459817886352 | accuracy: 77.19% | val_loss: 0.5548663316294551 | val_accuracy: 79.84%
[4, 180] loss: 0.4781560659408569 | accuracy: 83.44% | val_loss: 0.5309056667611003 | val_accuracy: 81.24%
[5, 10] loss: 0.5192959904670715 | accuracy: 78.75% | val_loss: 0.5601737322285771 | val_accuracy: 79.54%
```

```
[5, 20] loss: 0.4198275417089462 | accuracy: 85.00% | val_loss: 0.5875872587785125 | val_accuracy: 78.84%
[5, 30] loss: 0.5472079604864121 | accuracy: 78.12% | val_loss: 0.563591078389436 | val_accuracy: 78.84%
[5, 40] loss: 0.5152764856815338 | accuracy: 79.38% | val_loss: 0.5569407735019922 | val_accuracy: 79.34%
[5, 50] loss: 0.4879161223769188 | accuracy: 78.44% | val_loss: 0.5757614299654961 | val_accuracy: 78.04%
[5, 60] loss: 0.5253973752260208 | accuracy: 79.69% | val_loss: 0.5677738841623068 | val_accuracy: 80.64%
[5, 70] loss: 0.5185932159423828 | accuracy: 80.62% | val_loss: 0.5206415299326181 | val_accuracy: 80.74%
[5, 80] loss: 0.4829205870628357 | accuracy: 81.88% | val_loss: 0.5826818835921586 | val_accuracy: 79.74%
[5, 90] loss: 0.4180761307477951 | accuracy: 83.44% | val_loss: 0.5582622559741139 | val_accuracy: 80.34%
[5, 100] loss: 0.5678484737873077 | accuracy: 81.88% | val_loss: 0.5640545962378383 | val_accuracy: 79.94%
[5, 110] loss: 0.5328813463449478 | accuracy: 81.56% | val_loss: 0.5362056591548026 | val_accuracy: 81.04%
[5, 120] loss: 0.48887425661087036 | accuracy: 80.94% | val_loss: 0.5367208616808057 | val_accuracy: 79.84%
[5, 130] loss: 0.5255412548780442 | accuracy: 80.62% | val_loss: 0.5203068135306239 | val_accuracy: 81.54%
[5, 140] loss: 0.5337471157312393 | accuracy: 80.94% | val_loss: 0.5217169998213649 | val_accuracy: 81.24%
[5, 150] loss: 0.5608465760946274 | accuracy: 78.12% | val_loss: 0.5127564985305071 | val_accuracy: 81.84%
[5, 160] loss: 0.5229987412691116 | accuracy: 78.44% | val_loss: 0.5355646880343556 | val_accuracy: 81.24%
[5, 170] loss: 0.47168145328760147 | accuracy: 84.69% | val_loss: 0.5329043385572731 | val_accuracy: 81.34%
[5, 180] loss: 0.48969523161649703 | accuracy: 84.38% | val_loss: 0.5430097100324929 | val_accuracy: 79.64%
```

In [35]:
```python
# Get the number_to_label_array so that we can use it later to plot the confusion matrix
resnet_number_to_label_array = resnet_testloader.dataset.dataset.dataset.number_to_label_array
# Test the model
test_net(resnet_model, resnet_testloader,resnet_number_to_label_array, loss=resnet_loss, device=device)
```

```
Test loss: 0.5500709414482117 | Test accuracy: 79.73%
```
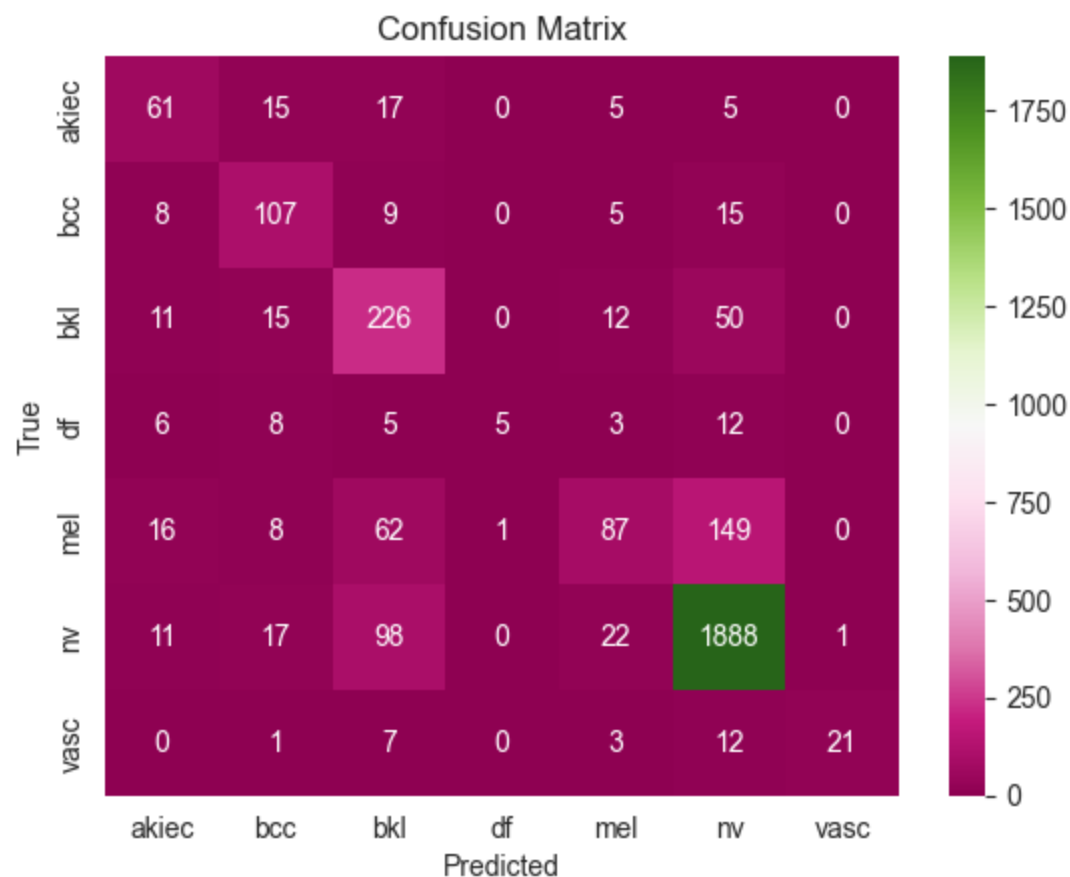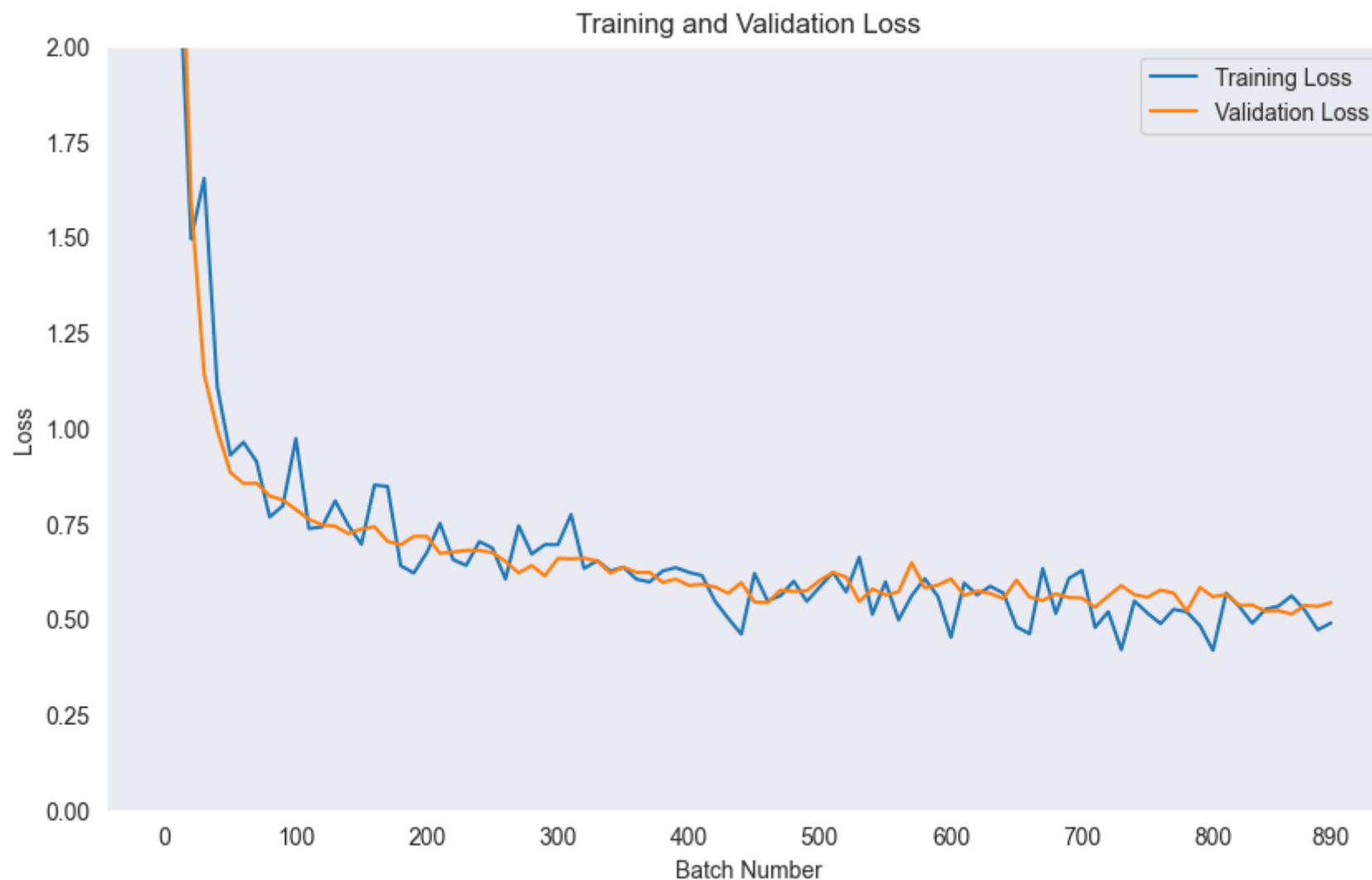
Observations

- Way better performance than the Simple and Complex models, huge recall and precision for all labels
- The `bkl` label is now the biggest misclassification, for the `nv` label, which is weird in comparison to our previous results
- The precession on the `nv` label is WAY better than the Simple and Complex models, which is a good thing, because the `nv` label is the most common label in the dataset

```
In [24]:  # Copying the lists so that I don't overwrite them
          resnet_train_loss = copy.deepcopy(train_loss)
          resnet_val_loss = copy.deepcopy(val_loss)
```

```
resnet_train_accuracy = copy.deepcopy(train_accuracy)
resnet_val_accuracy = copy.deepcopy(val_accuracy)
```

In [25]:
```
x_axis = list(map(lambda x:x*10,range(len(resnet_train_loss))))
plt.figure(figsize=(10,6))
plt.plot(x_axis ,resnet_train_loss, label='Training Loss')
plt.plot(x_axis , resnet_val_loss,label='Validation Loss')
xticks = np.arange(x_axis[0], x_axis[-1]+1, 100.0)
plt.xticks([*xticks, x_axis[-1]])
plt.ylim((0,2))
plt.xlabel('Batch Number')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.grid()
plt.show()
```

Observations

- The model is WAY less "jagged", which could be the result of the optimizer choice and settings or because the number of batches is way less than the Simple and Complex models
- The validation and training curves are very close together, which is a good thing, because the model is improving on both sets
- The model doesn't overfit

```python
In [26]: x_axis = list(map(lambda x:x*10,range(len(resnet_train_accuracy))))
         plt.figure(figsize=(10,6))
         plt.plot(x_axis ,resnet_train_accuracy, label='Training Accuracy')
         plt.plot(x_axis , resnet_val_accuracy,label='Validation Accuracy')
         xticks = np.arange(x_axis[0], x_axis[-1]+1, 100.0)
         plt.xticks([*xticks, x_axis[-1]])
         plt.xlabel('Batch Number')
         plt.ylabel('Accuracy')
         plt.title('Training and Validation Accuracy')
         plt.legend()
         plt.grid()
         plt.show()
```
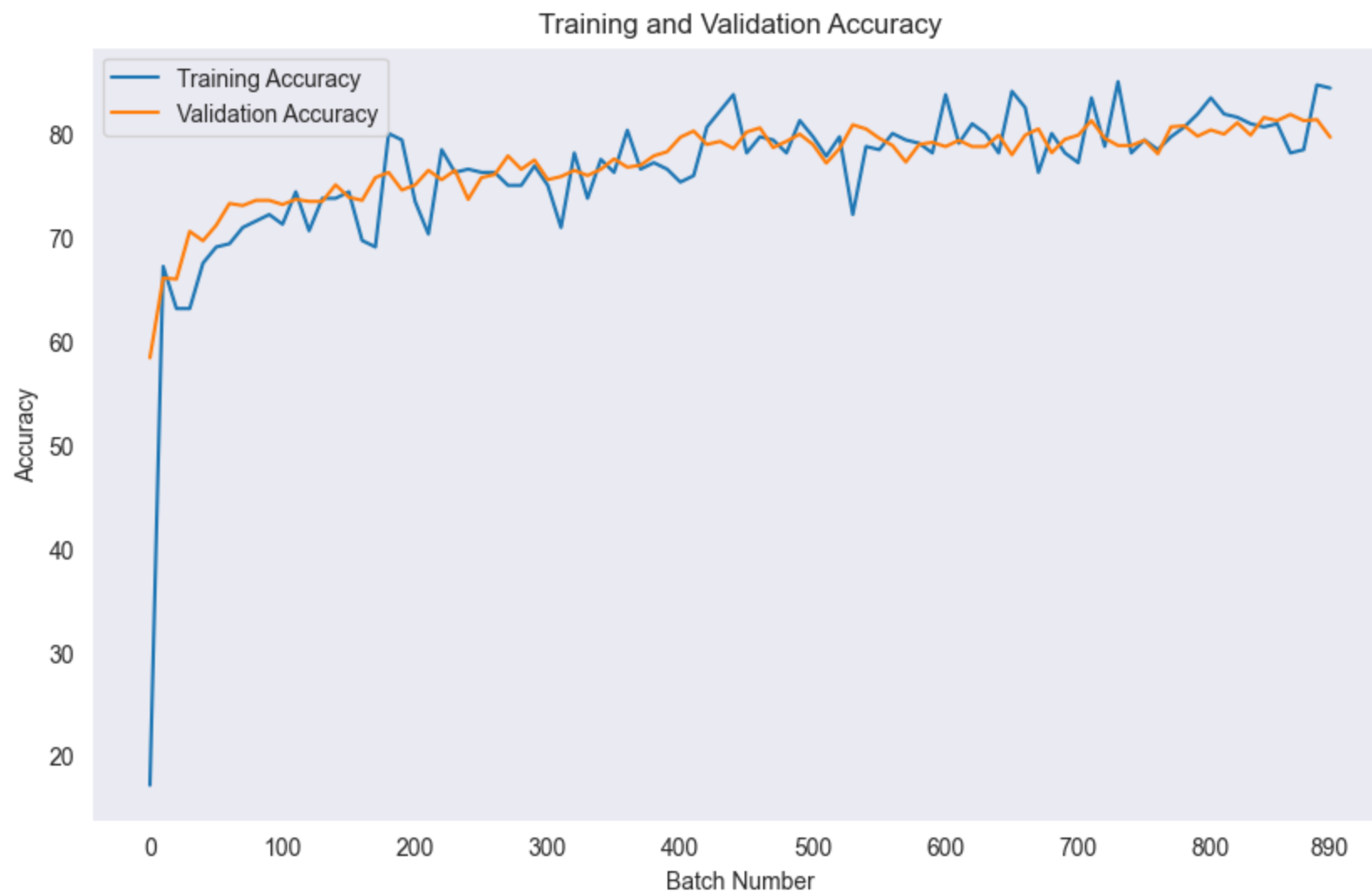
Observations

- Again, less "jagged" than the Simple and Complex models
- The model doesn't overfit
- The validation and training curves are very close together, which is a good thing, because the model is improving on both sets

- and something we can see here better than on the loss plot, is that the performance is better than the Simple and Complex models