Report

Date of Submission: 14-11-2024

Entry Number: 2024AIB2286, 2024AIB2287, 2024CSY7560, 2024CSY7558

Objective Function

The objective is to give only those predictions which are with high confidence and developing machine learning models that achieve **high overall accuracy** while ensuring **high-confidence predictions** using the CIFAR-100 dataset that was provided as part of the assignment.

Model Architecture:

We have tried multiple models using pytorch train and validation data so as to find test accuracy and check which model performs better. The model performing better in pytorch data was then later used for training on kaggle dataset.

Models used are as below:
efficientnet-B0, eefficientnet-B3, efficientnet-Lite, Enhanced CNN (custom CNN model), resnet-18, wide-resnet and other custom CNN models.

The epoch wise loss and test accuracy of models can be found in the later part of the report. It was observed that wide-resnet was converging to the better accuracy faster and as training data was small and 5 hours of training time so we went ahead with wide-resnet model

Approach used while training the model

Hyper parameter tuning (where parameters like batch size, learning rate and early stopping) was changed and accordingly accuracy was detected.

We tried Adam and multiple other optimizers and found that Adam is going ahead.

We tried schedulers like stepLR, cyclicLR, cosine annealing and so found that cyclicLR was performing better and hence went ahead with cyclicLR

We tried multiple loss functions from crossEntropyLoss, focalLoss and even tried making a custom loss function which gives 0 loss for correct classification and -5 loss for false prediction. Found that crossEntropyLoss outputs are better and give more prediction scores.

As the training dataset is small, We tried k-fold cross validation on the kaggle platform but it was taking too much time to train. As we had limited GPU hours available on kaggle, we decided not to use k-fold cross validation.

We also tried multiple data transformation like transforms.RandomCrop(32, padding=4),

transforms.RandomHorizontalFlip(), transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2), transforms.Normalize((0.5071, 0.4865, 0.4409), (0.2673, 0.2564, 0.2762)) etc. along with there permutation and combination.

We went ahead with random crop and random horizontal flip only as other transforms were making predictions tough.

We found that more data would be better as the model has reached its max and requires some more data or other data transformation so as to get better accuracy so as to do so, we added data augmentation transformation which increased the accuracy of model from 75% to 77%.

As the earlier 2 submissions were with penalty term as 0.7 and later was with 0.99. In Order to predict only when confidence is best, we also added temperature scaling and then threshold before prediction of the class.

Justification of Approach and Experiments:

As explained above We have tried different variations but decided to go with the architecture which was giving best results after training and optimising the same. We have attached the various variations tried and results. One of such experiment is given as below:

WRN-28-10 Training on CIFAR-100

We tested it out with two different loss functions – Cross-Entropy and Focal Loss – just to see what would happen. We used a WideResNet-28-10,. It's got 28 layers and 10 times wider than your usual ResNet . This architecture is known for doing pretty well with image classification tasks.

- **Training Details:** We trained for 150 epochs with a batch size of 128. We used the Adam optimizer with a learning rate of 0.01 and a scheduler to decrease the learning rate over time.

What We Found:

- **Training Loss:** Both loss functions did a good job of minimising the training loss. It dropped significantly in the beginning and then gradually levelled off.
  Cross-entropy reached a loss of 0.00021, while Focal Loss got down to 0.000069.
- **Time:** Each training run took about 5-6 hours to complete.

Things to Improve:

- To check validation accuracy
- **Try different parameters**
- **Hyperparameter Tuning**

- As the training dataset is small, We tried k-fold cross validation on kaggle platform but it was taking too much time to train. As we had limited GPU hours available on kaggle, we decided not to use k-fold cross validation
- We were not able to use various optimization techniques due to resource constraints such as GridSearch, Random Search or PSO. But the parameters have been found after multiple training sessions on different kaggle notebooks.

Use of Open Source Libraries, codes and Models

- Libraries and Tools: Torch, PIL, NumPy
- Weight Initialisations for models
- Additional Resources

Training

- **Data Preparation**
  The pickle file
- **Training Methodology**

Inference

- **Prediction Methodology:**
  If Predicted probability greater or equal to than 0.99 give the prediction else -1
- **Confidence Threshold** : 0.99

Calibration techniques:

As the training time is too much and due to lack of GPU resources. We could not try GridSearch, RandomSearch or Particle Swarm Optimization techniques for hyper parameters.

Instead all 4 team members used their free quota on kaggle platform to train the models on different parameters and compared and combined the results. Our Submission score started from -28000 and the latest kaggle score is more than 6400.

We have decided to write each finding in excel and communicate with a common whatsapp group.

Variations tried:

Type	Variations Tried
Model	efficientnet-B0, eefficientnet-B3, efficientnet-Lite, Enhanced CNN (custom CNN model), resnet-18,VGG-16,VGG-19,MobileNetV2, wide-resnet, resnext and other custom CNN models.
Optimizer	Adam, SGD
Loss Function	Cross Entropy, Penalty for wrong classification
Transforms	RandomCrop, HorizontalFlip, Colorjitter, RandomErasing,
Schedulers	StepLR, CyclicLR, CosineAnnealingLR

Pre processing Steps

Using the torch library transform sub module, we have applied RandomCrop and Random HorizontalFlip as the transform to the training data but the same has not been applied to the test model. We also added normalisation and data augmentation so as to increase accuracy and help models to learn features.

Post-processing Steps

Test model has not been applied to any transform apart from normalisation, which was applied when training.

Custom CNN model:

```
class DeeperEnhancedCNN(nn.Module):
    def __init__(self):
        super(DeeperEnhancedCNN, self).__init__()
        # First block
        self.conv1 = nn.Conv2d(3, 128, kernel_size=3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(128)
        self.conv2 = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(256)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Second block
        self.conv3 = nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=1)
        self.bn3 = nn.BatchNorm2d(512)
        self.conv4 = nn.Conv2d(512, 1024, kernel_size=3, stride=1, padding=1)
        self.bn4 = nn.BatchNorm2d(1024)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Third block - Added more layers here
        self.conv5 = nn.Conv2d(1024, 2048, kernel_size=3, stride=1, padding=1)
        self.bn5 = nn.BatchNorm2d(2048)
        self.conv6 = nn.Conv2d(2048, 4096, kernel_size=3, stride=1, padding=1)  # Extra
convolution
        self.bn6 = nn.BatchNorm2d(4096)
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.relu = nn.PReLU()
        self.dropout = nn.Dropout(p=0.4)

        # Adjusted the fully connected layer
        self.fc1 = nn.Linear(4096 * 4 * 4, 4096)
        self.fc2 = nn.Linear(4096, 1024)
        self.fc3 = nn.Linear(1024, 100)  # For CIFAR-100

    def forward(self, x):
        # First block
        x = self.pool1(self.relu(self.bn1(self.conv1(x))))
        x = self.relu(self.bn2(self.conv2(x)))

        # Second block
        x = self.pool2(self.relu(self.bn3(self.conv3(x))))
        x = self.relu(self.bn4(self.conv4(x)))

        # Third block
```

```
        x = self.pool3(self.relu(self.bn5(self.conv5(x))))
        x = self.relu(self.bn6(self.conv6(x)))

        # Flatten the tensor
        x = x.view(x.size(0), -1)

        # Fully connected layers
        x = self.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(x)

        return x
```

The above model was ran with batch size 128 till 100 epochs and max accuracy with test set was found to be 57%.

Wideresnet:

```
# Define the Basic Block used in WideResNet
class BasicBlock(nn.Module):
    def __init__(self, in_planes, out_planes, stride, drop_rate=0.3):
    super(BasicBlock, self).__init__()
    self.bn1 = nn.BatchNorm2d(in_planes)
    self.relu = nn.ReLU(inplace=True)
    self.conv1 = nn.Conv2d(in_planes, out_planes, kernel_size=3, stride=stride,
padding=1, bias=False)
    self.bn2 = nn.BatchNorm2d(out_planes)
    self.conv2 = nn.Conv2d(out_planes, out_planes, kernel_size=3, stride=1, padding=1,
bias=False)
    self.drop_rate = drop_rate
    self.equalInOut = (in_planes == out_planes)
    self.convShortcut = (not self.equalInOut) and nn.Conv2d(in_planes, out_planes,
kernel_size=1, stride=stride, padding=0, bias=False) or None

    def forward(self, x):
    out = self.relu(self.bn1(x))
    if not self.equalInOut:
    x = out
    out = self.relu(self.bn2(self.conv1(out)))
    if self.drop_rate > 0:
    out = F.dropout(out, p=self.drop_rate, training=self.training)
```

```
        out = self.conv2(out)
        return torch.add(x if self.convShortcut is None else self.convShortcut(x), out)

# Define Network Block
class NetworkBlock(nn.Module):
        def __init__(self, nb_layers, in_planes, out_planes, block, stride, drop_rate=0.3):
        super(NetworkBlock, self).__init__()
        self.layer = self._make_layer(block, in_planes, out_planes, nb_layers, stride,
drop_rate)

        def _make_layer(self, block, in_planes, out_planes, nb_layers, stride, drop_rate):
        layers = []
        for i in range(nb_layers):
        layers.append(block(i == 0 and in_planes or out_planes, out_planes, i == 0 and
stride or 1, drop_rate))
        return nn.Sequential(*layers)

        def forward(self, x):
        return self.layer(x)

# Define the WideResNet Model
class WideResNet(nn.Module):
        def __init__(self, depth, num_classes, widen_factor=1, drop_rate=0.3):
        super(WideResNet, self).__init__()
        self.in_planes = 16
        assert (depth - 4) % 6 == 0, 'Depth should be 6n+4'
        n = (depth - 4) // 6
        k = widen_factor
        nStages = [16, 16 * k, 32 * k, 64 * k]

        self.conv1 = nn.Conv2d(3, nStages[0], kernel_size=3, stride=1, padding=1,
bias=False)
        self.block1 = NetworkBlock(n, nStages[0], nStages[1], BasicBlock, 1, drop_rate)
        self.block2 = NetworkBlock(n, nStages[1], nStages[2], BasicBlock, 2, drop_rate)
        self.block3 = NetworkBlock(n, nStages[2], nStages[3], BasicBlock, 2, drop_rate)
        self.bn1 = nn.BatchNorm2d(nStages[3])
        self.relu = nn.ReLU(inplace=True)
        self.fc = nn.Linear(nStages[3], num_classes)
        self.drop_rate = drop_rate

        for m in self.modules():
        if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')

        def forward(self, x):
        out = self.conv1(x)
        out = self.block1(out)
        out = self.block2(out)
```

```
        out = self.block3(out)
        out = self.relu(self.bn1(out))
        out = F.avg_pool2d(out, 8)
        out = out.view(-1, self.fc.in_features)
        return self.fc(out)
```

The above model was ran with 128 batch size and till convergence. Below is the loss and accuracy with test set observed

wideresnet model:
Epoch 1/500, Loss: 3.8958370655089083, accuracy: 13.39%Epoch 2/500, Loss: 3.1907375718626523, accuracy: 23.90%Epoch 3/500, Loss: 2.66034812207722, accuracy: 33.61%Epoch 4/500, Loss: 2.2760456333989683, accuracy: 39.56%Epoch 5/500, Loss: 1.980759484078878, accuracy: 42.98%Epoch 6/500, Loss: 1.75116037956589, accuracy: 46.99%Epoch 7/500, Loss: 1.5703523268785013, accuracy: 51.47%Epoch 8/500, Loss: 1.416255613727033, accuracy: 53.48%Epoch 9/500, Loss: 1.2821660611940466, accuracy: 56.87%Epoch 10/500, Loss: 1.1641356271246206, accuracy: 56.43%Epoch 11/500, Loss: 1.053151973982906, accuracy: 57.16%Epoch 12/500, Loss: 0.9569692506509668, accuracy: 58.58%Epoch 13/500, Loss: 0.8613696657788114, accuracy: 60.20%Epoch 14/500, Loss: 0.7829350902296394, accuracy: 59.80%Epoch 15/500, Loss: 0.7034277122496339, accuracy: 60.35%Epoch 16/500, Loss: 0.6365693751198557, accuracy: 61.36%Epoch 17/500, Loss: 0.5702269039190638, accuracy: 62.70%Epoch 18/500, Loss: 0.5269172452294918, accuracy: 59.15%Epoch 19/500, Loss: 0.47140407493657166, accuracy: 61.18%Epoch 20/500, Loss: 0.42413251654571277, accuracy: 62.86%Epoch 21/500, Loss: 0.3901892089858994, accuracy: 61.83%Epoch 22/500, Loss: 0.3585834849597243, accuracy: 60.09%Epoch 23/500, Loss: 0.3293017967963767, accuracy: 62.20%

Resnet 18:

model = models.resnet18(pretrained=False)

The model output where observed as below:
Epoch 1/100, Loss: 4.194681015160993
Validation Accuracy: 11.22%
Epoch 2/100, Loss: 3.710814584551565
Validation Accuracy: 16.08%
Epoch 3/100, Loss: 3.4643728909894937
Validation Accuracy: 18.89%
Epoch 4/100, Loss: 3.2846149745804576
Validation Accuracy: 21.70%

Epoch 5/100, Loss: 3.113571018209238
Validation Accuracy: 23.64%
Epoch 6/100, Loss: 2.9739636545595913
Validation Accuracy: 26.10%
Epoch 7/100, Loss: 2.8548560099833455
Validation Accuracy: 27.69%
Epoch 8/100, Loss: 2.743235691124216
Validation Accuracy: 30.38%
Epoch 9/100, Loss: 2.647169861342291
Validation Accuracy: 31.57%
Epoch 10/100, Loss: 2.569054919435545
Validation Accuracy: 32.60%
Epoch 11/100, Loss: 2.4967064083079853
Validation Accuracy: 33.25%
Epoch 12/100, Loss: 2.4184513579853966
Validation Accuracy: 34.98%
Epoch 13/100, Loss: 2.3558037220059758
Validation Accuracy: 35.83%
Epoch 14/100, Loss: 2.2940699980996757
Validation Accuracy: 36.07%
Epoch 15/100, Loss: 2.2353271904503904
Validation Accuracy: 37.40%
Epoch 16/100, Loss: 2.1800304642113884
Validation Accuracy: 37.41%
Epoch 17/100, Loss: 2.1265135479095343
Validation Accuracy: 38.73%
Epoch 18/100, Loss: 2.0913470681670985
Validation Accuracy: 38.61%
Epoch 19/100, Loss: 2.03326692666544
Validation Accuracy: 38.92%
Epoch 20/100, Loss: 1.9872566513393237
Validation Accuracy: 39.70%
Epoch 21/100, Loss: 1.9433395149152908
Validation Accuracy: 40.77%
Epoch 22/100, Loss: 1.8978181278614132
Validation Accuracy: 41.37%
Epoch 23/100, Loss: 1.8598604290686604
Validation Accuracy: 41.42%
Epoch 24/100, Loss: 1.813386885101533
Validation Accuracy: 41.48%
Epoch 25/100, Loss: 1.772424898793935
Validation Accuracy: 41.60%
Epoch 26/100, Loss: 1.7409106005183266
Validation Accuracy: 42.30%
Epoch 27/100, Loss: 1.6968459147016715
Validation Accuracy: 43.67%
Epoch 28/100, Loss: 1.6643307340114624
Validation Accuracy: 43.05%

Epoch 29/100, Loss: 1.6227991324861337
Validation Accuracy: 42.57%
Epoch 30/100, Loss: 1.589911506913812
Validation Accuracy: 43.39%
Early stopping triggered.
Test Accuracy: 42.56%
Execution time: 959.2236661911011

Efficientnetlite:

```
class EfficientNetLiteCNN(nn.Module):
    def __init__(self):
        super(EfficientNetLiteCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1, stride=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.dwconv2 = nn.Conv2d(32, 32, kernel_size=3, padding=1, groups=32)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=1)
        self.bn2 = nn.BatchNorm2d(64)
        self.dwconv3 = nn.Conv2d(64, 64, kernel_size=3, padding=1, groups=64)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=1)
        self.bn3 = nn.BatchNorm2d(128)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(128 * 8 * 8, 512)
        self.dropout = nn.Dropout(0.5)
        self.fc2 = nn.Linear(512, 100)

    def forward(self, x):
        x = torch.relu(self.bn1(self.conv1(x)))
        x = torch.relu(self.bn2(self.conv2(self.dwconv2(x))))
        x = self.pool(x)
        x = torch.relu(self.bn3(self.conv3(self.dwconv3(x))))
        x = self.pool(x)
        x = x.view(-1, 128 * 8 * 8)
        x = torch.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x

# Hyperparameters
batch_size = 128
learning_rate = 0.001
```

Below are the observation found:

efficient net lite model:

Epoch 1/500, Loss: 4.360215240732178, accuracy: 10.19%Epoch 2/500, Loss: 4.027833084925971, accuracy: 13.71%Epoch 3/500, Loss: 3.8574972786866795, accuracy: 15.93%Epoch 4/500, Loss: 3.739587589907829, accuracy: 17.53%Epoch 5/500, Loss: 3.6378826143796488, accuracy: 19.27%Epoch 6/500, Loss: 3.5537267566641884, accuracy: 21.19%Epoch 7/500, Loss: 3.476924180374731, accuracy: 21.70%Epoch 8/500, Loss: 3.408281158608244, accuracy: 23.11%Epoch 9/500, Loss: 3.340460462033596, accuracy: 24.61%Epoch 10/500, Loss: 3.286372683542159, accuracy: 25.66%Epoch 11/500, Loss: 3.228576555276466, accuracy: 26.80%Epoch 12/500, Loss: 3.172423386512815, accuracy: 27.75%Epoch 13/500, Loss: 3.138412274363096, accuracy: 28.80%Epoch 14/500, Loss: 3.1039964268579507, accuracy: 28.94%Epoch 15/500, Loss: 3.065286183296262, accuracy: 29.72%Epoch 16/500, Loss: 3.0393613207980494, accuracy: 30.62%Epoch 17/500, Loss: 3.015606296031981, accuracy: 30.00%Epoch 18/500, Loss: 2.986381257586467, accuracy: 30.73%Epoch 19/500, Loss: 2.968880922288236, accuracy: 31.91%Epoch 20/500, Loss: 2.946484629760313, accuracy: 32.48%Epoch 21/500, Loss: 2.9324763790725745, accuracy: 32.38%Epoch 22/500, Loss: 2.91599942229288, accuracy: 32.97%Epoch 23/500, Loss: 2.891361853655647, accuracy: 33.38%Epoch 24/500, Loss: 2.878907850636241, accuracy: 32.76%Epoch 25/500, Loss: 2.862930344803559, accuracy: 34.01%Epoch 26/500, Loss: 2.8489256868581943, accuracy: 33.72%Epoch 27/500, Loss: 2.836324024078486, accuracy: 34.62%Epoch 28/500, Loss: 2.827077588766737, accuracy: 35.48%Epoch 29/500, Loss: 2.800273266594733, accuracy: 35.27%Epoch 30/500, Loss: 2.799769186302829, accuracy: 35.07%Epoch 31/500, Loss: 2.7807259108404367, accuracy: 35.97%Epoch 32/500, Loss: 2.7779379732468548, accuracy: 35.48%Epoch 33/500, Loss: 2.7674035788192164, accuracy: 36.27%Epoch 34/500, Loss: 2.7540074364303626, accuracy: 37.04%Epoch 35/500, Loss: 2.7438957666801977, accuracy: 36.99%Epoch 36/500, Loss: 2.726191955454209, accuracy: 36.47%Epoch 37/500, Loss: 2.729940025397884, accuracy: 36.88%Epoch 38/500, Loss: 2.7138825700715983, accuracy: 37.44%Epoch 39/500, Loss: 2.710140200832006, accuracy: 36.63%Epoch 40/500, Loss: 2.694559768642611, accuracy: 37.98%Epoch 41/500, Loss: 2.6931150307130935, accuracy: 38.27%Epoch 42/500, Loss: 2.6897601581290553, accuracy: 37.46%Epoch 43/500, Loss: 2.6791635638917497, accuracy: 38.33%Epoch 44/500, Loss: 2.657219436772339, accuracy: 38.35%Epoch 45/500, Loss: 2.6526657004490533, accuracy: 38.09%Epoch 46/500, Loss: 2.6493101705370656, accuracy: 38.96%Epoch 47/500, Loss: 2.6280903041820087, accuracy: 38.76%Epoch 48/500, Loss: 2.628377166855366, accuracy: 39.72%Epoch 49/500, Loss: 2.6115937623221552, accuracy: 38.53%Epoch 50/500, Loss: 2.614183448464669, accuracy: 40.15%Epoch 51/500, Loss: 2.6038029352417382, accuracy: 38.83%Epoch 52/500, Loss: 2.6081756447892053, accuracy: 40.55%Epoch 53/500, Loss: 2.5963790099639112, accuracy: 39.75%Epoch 54/500, Loss: 2.582945385247545, accuracy: 40.82%Epoch 55/500, Loss: 2.579683523348835, accuracy: 40.78%Epoch 56/500, Loss: 2.5666697872874074, accuracy: 40.06%Epoch 57/500, Loss: 2.5633385998513694, accuracy: 39.17%Epoch 58/500, Loss: 2.559815848270036, accuracy: 41.18%Epoch 59/500, Loss: 2.5505428295916, accuracy: 40.18%Epoch 60/500, Loss: 2.540532194440017, accuracy: 40.55%Epoch 61/500, Loss: 2.528377365883049, accuracy: 41.00%Epoch 62/500, Loss: 2.527028257584633, accuracy: 40.89%Epoch 63/500, Loss: 2.5111712324040014, accuracy: 41.87%Epoch 64/500, Loss: 2.515386178365449, accuracy: 41.88%Epoch 65/500, Loss: 2.5021812751165133, accuracy: 41.49%Epoch 66/500, Loss: 2.493788082581347, accuracy: 41.70%Epoch 67/500, Loss: 2.5010043611306974, accuracy: 41.47%Epoch 68/500, Loss: 2.4815885093815795, accuracy: 41.64%Epoch 69/500, Loss: 2.475962230311635, accuracy: 42.38%Epoch 70/500, Loss: 2.471860908181466, accuracy: 41.34%Epoch 71/500, Loss: 2.448628339011346, accuracy: 41.62%Epoch 72/500, Loss: 2.4667965438969603, accuracy: 40.74%Epoch 73/500, Loss: 2.462963899383155, accuracy: 42.62%Epoch 74/500, Loss: 2.4485643702699704, accuracy: 42.27%Epoch 75/500, Loss: 2.437971852929391, accuracy: 42.93%Epoch 76/500, Loss: 2.4348993405051855, accuracy: 43.14%Epoch 77/500, Loss: 2.4318822302171945, accuracy: 42.63%Epoch 78/500, Loss: 2.428695847311288, accuracy: 42.56%Epoch 79/500, Loss: 2.416344038970635, accuracy: 42.63%Epoch 80/500, Loss: 2.4110349538686027, accuracy: 42.75%Epoch 81/500, Loss: 2.4078052007328825, accuracy: 43.66%Epoch 82/500, Loss: 2.392879025710513, accuracy: 42.95%Epoch 83/500, Loss: 2.3909472004531898, accuracy: 43.54%Epoch 84/500, Loss: 2.3889439816365154, accuracy: 42.57%Epoch 85/500, Loss: 2.3828033955810626, accuracy: 43.94%Epoch 86/500, Loss: 2.3701141920236064, accuracy: 43.97%Epoch 87/500, Loss: 2.372822029511337, accuracy: 43.98%Epoch 88/500, Loss: 2.3637990914952116, accuracy: 43.31%Epoch 89/500, Loss: 2.3598635376566817, accuracy: 43.09%Epoch 90/500, Loss: 2.3472682024206954, accuracy: 44.15%Epoch 91/500, Loss: 2.360951248947007, accuracy: 43.62%Epoch 92/500, Loss: 2.3421943681624233, accuracy: 43.96%Epoch 93/500, Loss: 2.329005687742892, accuracy: 44.26%Epoch 94/500, Loss: 2.332755136367915, accuracy: 44.56%Epoch 95/500, Loss: 2.3276535437235135, accuracy: 45.47%Epoch 96/500, Loss: 2.319227936932498, accuracy: 44.42%Epoch 97/500, Loss: 2.31093358078881, accuracy: 45.11%Epoch 98/500, Loss: 2.3100625118026343, accuracy: 44.93%Epoch 99/500, Loss: 2.313205302218952, accuracy: 45.06%Epoch 100/500, Loss: 2.309698928652517, accuracy: 44.90%Epoch 101/500, Loss: 2.2937696568496393, accuracy: 44.80%Epoch 102/500, Loss: 2.2878667171044116, accuracy: 45.31%Epoch 103/500, Loss: 2.2849646065851004, accuracy: 44.64%Epoch 104/500, Loss: 2.279021483248152, accuracy: 44.89%Epoch 105/500, Loss: 2.2824555348862163, accuracy: 43.38%

MobileNet - v2:

model = models.mobilenet_v2(pretrained=False)

Below are the observation found for model:

Epoch 1/500, Loss: 4.603856518445418, accuracy: 2.09%Epoch 2/500, Loss: 4.514468612573336, accuracy: 2.59%Epoch 3/500, Loss: 4.448392982678035, accuracy: 3.54%Epoch 4/500, Loss: 4.381721428288218, accuracy: 3.92%Epoch 5/500, Loss: 4.297346371214103, accuracy: 5.24%Epoch 6/500, Loss: 4.217047803542194, accuracy: 6.52%Epoch 7/500, Loss: 4.138686625244063, accuracy: 7.18%Epoch 8/500, Loss: 4.067462747968981, accuracy: 7.89%Epoch 9/500, Loss: 4.007351367979708, accuracy: 9.16%Epoch 10/500, Loss: 3.9429921195330215, accuracy: 9.99%Epoch 11/500, Loss: 3.8775348120638173, accuracy: 10.70%Epoch 12/500, Loss: 3.816184785359961, accuracy: 12.39%Epoch 13/500, Loss: 3.76094287252792, accuracy: 12.99%Epoch 14/500, Loss: 3.6912881443872476, accuracy: 14.29%Epoch 15/500, Loss: 3.634215004913642, accuracy: 15.19%Epoch 16/500, Loss: 3.5748970069543784, accuracy: 15.80%Epoch

17/500, Loss: 3.522492667293305, accuracy: 16.89%Epoch 18/500, Loss: 3.4652341271910214, accuracy: 18.09%Epoch 19/500, Loss: 3.406921383669919, accuracy: 18.35%Epoch 20/500, Loss: 3.361202163769461, accuracy: 19.79%Epoch 21/500, Loss: 3.3026282153166164, accuracy: 20.46%Epoch 22/500, Loss: 3.265655853559294, accuracy: 20.59%Epoch 23/500, Loss: 3.2140242779041497, accuracy: 21.65%Epoch 24/500, Loss: 3.1664164480955703, accuracy: 22.41%Epoch 25/500, Loss: 3.1225082666977593, accuracy: 23.15%Epoch 26/500, Loss: 3.0812546566624164, accuracy: 23.88%Epoch 27/500, Loss: 3.039326575101184, accuracy: 24.92%Epoch 28/500, Loss: 3.001599287437966, accuracy: 24.49%Epoch 29/500, Loss: 2.9614796918981217, accuracy: 25.05%Epoch 30/500, Loss: 2.91814642306179, accuracy: 26.19%Epoch 31/500, Loss: 2.8867737850569704, accuracy: 26.49%Epoch 32/500, Loss: 2.8578186968098516, accuracy: 27.29%Epoch 33/500, Loss: 2.814201757426152, accuracy: 26.95%Epoch 34/500, Loss: 2.7827519455834118, accuracy: 28.35%Epoch 35/500, Loss: 2.751336758703832, accuracy: 28.32%Epoch 36/500, Loss: 2.7173804120944283, accuracy: 29.62%Epoch 37/500, Loss: 2.6782970983354026, accuracy: 29.43%Epoch 38/500, Loss: 2.6444375984504096, accuracy: 30.63%Epoch 39/500, Loss: 2.612551815979316, accuracy: 30.03%Epoch 40/500, Loss: 2.575558686195432, accuracy: 31.12%Epoch 41/500, Loss: 2.544057719237969, accuracy: 32.37%Epoch 42/500, Loss: 2.5215466906652426, accuracy: 31.63%Epoch 43/500, Loss: 2.4730424850493136, accuracy: 32.76%Epoch 44/500, Loss: 2.4589860366128593, accuracy: 32.54%Epoch 45/500, Loss: 2.4271067194926466, accuracy: 34.29%Epoch 46/500, Loss: 2.3912697716442217, accuracy: 33.97%Epoch 47/500, Loss: 2.35498140504598, accuracy: 34.12%Epoch 48/500, Loss: 2.328483898316503, accuracy: 34.36%Epoch 49/500, Loss: 2.304700252650034, accuracy: 34.99%Epoch 50/500, Loss: 2.272432833681326, accuracy: 35.50%Epoch 51/500, Loss: 2.239185177151809, accuracy: 35.47%Epoch 52/500, Loss: 2.21773415971595, accuracy: 35.42%Epoch 53/500, Loss: 2.1925170692946296, accuracy: 36.16%Epoch 54/500, Loss: 2.1667599147542966, accuracy: 36.45%Epoch 55/500, Loss: 2.1440669779887287, accuracy: 37.13%Epoch 56/500, Loss: 2.1143350162164634, accuracy: 37.11%Epoch 57/500, Loss: 2.087872126218303, accuracy: 37.48%Epoch 58/500, Loss: 2.067683346436152, accuracy: 37.27%Epoch 59/500, Loss: 2.038112694040284, accuracy: 37.89%Epoch 60/500, Loss: 2.028090117532579, accuracy: 37.48%Epoch 61/500, Loss: 1.982333795798709, accuracy: 38.56%Epoch 62/500, Loss: 1.9776658213047116, accuracy: 38.03%Epoch 63/500, Loss: 1.9379768063650107, accuracy: 37.43%Epoch 64/500, Loss: 1.920553028431085, accuracy: 39.05%Epoch 65/500, Loss: 1.9018781310152215, accuracy: 38.78%Epoch 66/500, Loss: 1.8866547477214843, accuracy: 38.56%Epoch 67/500, Loss: 1.8526559394338857, accuracy: 39.93%Epoch 68/500, Loss: 1.830819897334594, accuracy: 39.68%Epoch 69/500, Loss: 1.8184470142549871, accuracy: 38.97%Epoch 70/500, Loss: 1.8042222505335308, accuracy: 39.74%wide net model:

Epoch 1/500, Loss: 4.647676211793709, accuracy: 1.49%Epoch 2/500, Loss: 4.585302051680777, accuracy: 2.22%Epoch 3/500, Loss: 4.4861460573533005, accuracy: 3.53%Epoch 4/500, Loss: 4.386408864384722, accuracy: 4.27%Epoch 5/500, Loss: 4.3019383837804765, accuracy: 5.48%Epoch 6/500, Loss: 4.235437116354627, accuracy: 5.74%Epoch 7/500, Loss: 4.152704055961745, accuracy: 6.83%Epoch 8/500, Loss: 4.084733406905933, accuracy: 8.24%Epoch 9/500, Loss: 4.011959139343418, accuracy: 9.46%Epoch 10/500, Loss: 3.9428267594798445, accuracy: 10.50%Epoch 11/500, Loss:

3.8791943767186625, accuracy: 11.37%Epoch 12/500, Loss: 3.8155172999252747, accuracy: 11.90%Epoch 13/500, Loss: 3.7498889134058255, accuracy: 13.78%Epoch 14/500, Loss: 3.6886627454586955, accuracy: 14.12%Epoch 15/500, Loss: 3.6257643724036646, accuracy: 15.09%Epoch 16/500, Loss: 3.5627404149535975, accuracy: 16.13%Epoch 17/500, Loss: 3.5164357018287835, accuracy: 16.70%Epoch 18/500, Loss: 3.454591001086223, accuracy: 17.46%Epoch 19/500, Loss: 3.403981969789471, accuracy: 18.15%Epoch 20/500, Loss: 3.3558685456395456, accuracy: 20.08%Epoch 21/500, Loss: 3.307169325821235, accuracy: 19.82%Epoch 22/500, Loss: 3.254223558908838, accuracy: 21.33%Epoch 23/500, Loss: 3.2058057583811337, accuracy: 21.91%Epoch 24/500, Loss: 3.1551490238560436, accuracy: 22.24%Epoch 25/500, Loss: 3.1076970429676574, accuracy: 23.37%Epoch 26/500, Loss: 3.05836103212498, accuracy: 23.73%Epoch 27/500, Loss: 3.0109008139051743, accuracy: 25.46%Epoch 28/500, Loss: 2.969358307626241, accuracy: 25.02%Epoch 29/500, Loss: 2.9369211202997074, accuracy: 25.93%Epoch 30/500, Loss: 2.8919867963132346, accuracy: 26.40%Epoch 31/500, Loss: 2.853459740538731, accuracy: 27.49%Epoch 32/500, Loss: 2.808312281318333, accuracy: 27.97%Epoch 33/500, Loss: 2.770021930679946, accuracy: 29.48%Epoch 34/500, Loss: 2.730112189222175, accuracy: 29.47%Epoch 35/500, Loss: 2.692394308421923, accuracy: 29.37%Epoch 36/500, Loss: 2.656702952006894, accuracy: 29.71%Epoch 37/500, Loss: 2.621962042415843, accuracy: 30.59%Epoch 38/500, Loss: 2.5940499921588946, accuracy: 31.94%Epoch 39/500, Loss: 2.5587541672884657, accuracy: 32.00%Epoch 40/500, Loss: 2.528397195479449, accuracy: 32.04%Epoch 41/500, Loss: 2.4970059760696137, accuracy: 32.56%Epoch 42/500, Loss: 2.4571884349179083, accuracy: 33.42%Epoch 43/500, Loss: 2.4334481146634386, accuracy: 33.37%Epoch 44/500, Loss: 2.40537991121297, accuracy: 34.35%Epoch 45/500, Loss: 2.3839553472635995, accuracy: 34.58%Epoch 46/500, Loss: 2.3472972480232452, accuracy: 33.78%Epoch 47/500, Loss: 2.3199837909025303, accuracy: 34.57%Epoch 48/500, Loss: 2.29955858068393, accuracy: 35.14%Epoch 49/500, Loss: 2.2767575305441152, accuracy: 35.13%Epoch 50/500, Loss: 2.248543215529693, accuracy: 35.72%Epoch 51/500, Loss: 2.2193917411062722, accuracy: 35.49%Epoch 52/500, Loss: 2.195910219951054, accuracy: 36.07%Epoch 53/500, Loss: 2.1663334205022555, accuracy: 36.85%Epoch 54/500, Loss: 2.149907591702688, accuracy: 36.19%Epoch 55/500, Loss: 2.126648876063354, accuracy: 37.51%Epoch 56/500, Loss: 2.108421998255698, accuracy: 37.44%Epoch 57/500, Loss: 2.080849638985246, accuracy: 37.79%Epoch 58/500, Loss: 2.045118176723685, accuracy: 37.86%Epoch 59/500, Loss: 2.0330254848655835, accuracy: 37.92%Epoch 60/500, Loss: 2.0049837852073145, accuracy: 37.80%Epoch 61/500, Loss: 1.99305193655936, accuracy: 38.33%Epoch 62/500, Loss: 1.9649440870260644, accuracy: 37.84%Epoch 63/500, Loss: 1.9324648063201124, accuracy: 38.62%Epoch 64/500, Loss: 1.9221781644674822, accuracy: 39.20%Epoch 65/500, Loss: 1.9071738314445672, accuracy: 38.96%Epoch 66/500, Loss: 1.8741448565822123, accuracy: 39.23%Epoch 67/500, Loss: 1.8644547151482624, accuracy: 39.30%Epoch 68/500, Loss: 1.8341989419649325, accuracy: 39.04%Epoch 69/500, Loss: 1.810324299670851, accuracy: 39.13%Epoch 70/500, Loss: 1.7878377696742183, accuracy: 38.95%

Custom CNN:

model = Sequential()
# 128 and not only 32 filters because there are 100 classes. 32 filters gave bad results.
model.add(Conv2D(128, (3, 3), padding='same', input_shape=X_train.shape[1:]))
model.add(Activation('elu'))

model.add(Conv2D(128, (3, 3)))
model.add(Activation('elu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256, (3, 3), padding='same'))
model.add(Activation('elu'))

model.add(Conv2D(256, (3, 3)))
model.add(Activation('elu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(512, (3, 3), padding='same'))
model.add(Activation('elu'))

model.add(Conv2D(512, (3, 3)))
model.add(Activation('elu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(1024))
model.add(Activation('elu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))

model.summary()

Below are the observation found:

1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.2158 - val_loss: 3.3008
Epoch 3/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 40s 39ms/step - accuracy: 0.1876 - loss: 3.4240 - val_accuracy: 0.2981 - val_loss: 2.8770
Epoch 4/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.2981 - val_loss: 2.8770

Epoch 5/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 40s 39ms/step - accuracy: 0.2525 - loss: 3.0732 - val_accuracy: 0.3439 - val_loss: 2.6524
Epoch 6/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.3439 - val_loss: 2.6524
Epoch 7/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.2902 - loss: 2.8694 - val_accuracy: 0.3646 - val_loss: 2.5216
Epoch 8/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.3646 - val_loss: 2.5216
Epoch 9/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 40s 39ms/step - accuracy: 0.3290 - loss: 2.7072 - val_accuracy: 0.3970 - val_loss: 2.3511
Epoch 10/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.3970 - val_loss: 2.3511
Epoch 11/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.3521 - loss: 2.5788 - val_accuracy: 0.4164 - val_loss: 2.2587
Epoch 12/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.4164 - val_loss: 2.2587
Epoch 13/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 40s 39ms/step - accuracy: 0.3744 - loss: 2.4752 - val_accuracy: 0.4413 - val_loss: 2.1648
Epoch 14/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.4413 - val_loss: 2.1648
Epoch 15/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.3833 - loss: 2.4176 - val_accuracy: 0.4594 - val_loss: 2.1087
Epoch 16/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.4594 - val_loss: 2.1087
Epoch 17/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 40s 39ms/step - accuracy: 0.4028 - loss: 2.3433 - val_accuracy: 0.4738 - val_loss: 2.0255
Epoch 18/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.4738 - val_loss: 2.0255
Epoch 19/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.4170 - loss: 2.2627 - val_accuracy: 0.4769 - val_loss: 1.9993
Epoch 20/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.4769 - val_loss: 1.9993

Epoch 21/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 40s 40ms/step - accuracy: 0.4273 - loss: 2.2136 - val_accuracy: 0.4872 - val_loss: 1.9420
Epoch 22/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.4872 - val_loss: 1.9420
Epoch 23/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.4411 - loss: 2.1559 - val_accuracy: 0.5033 - val_loss: 1.8873
Epoch 24/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.5033 - val_loss: 1.8873
Epoch 25/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.4508 - loss: 2.1096 - val_accuracy: 0.5109 - val_loss: 1.8578
Epoch 26/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.5109 - val_loss: 1.8578
Epoch 27/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 40s 40ms/step - accuracy: 0.4566 - loss: 2.0708 - val_accuracy: 0.5055 - val_loss: 1.8593
Epoch 28/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.5055 - val_loss: 1.8593
Epoch 29/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.4742 - loss: 2.0049 - val_accuracy: 0.5154 - val_loss: 1.8104
Epoch 30/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.5154 - val_loss: 1.8104
Epoch 31/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.4782 - loss: 1.9715 - val_accuracy: 0.5338 - val_loss: 1.7592
Epoch 32/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.5338 - val_loss: 1.7592
Epoch 33/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.4887 - loss: 1.9317 - val_accuracy: 0.5389 - val_loss: 1.7319
Epoch 34/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.5389 - val_loss: 1.7319
Epoch 35/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.4923 - loss: 1.9115 - val_accuracy: 0.5450 - val_loss: 1.6767
Epoch 36/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.5450 - val_loss: 1.6767

Epoch 37/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.5006 - loss: 1.8671 - val_accuracy: 0.5426 - val_loss: 1.6868
Epoch 38/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.5426 - val_loss: 1.6868
Epoch 39/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.5052 - loss: 1.8473 - val_accuracy: 0.5504 - val_loss: 1.6775
Epoch 40/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.5504 - val_loss: 1.6775
Epoch 41/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 40s 40ms/step - accuracy: 0.5164 - loss: 1.7969 - val_accuracy: 0.5577 - val_loss: 1.6446
Epoch 42/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.5577 - val_loss: 1.6446
Epoch 43/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.5212 - loss: 1.7725 - val_accuracy: 0.5623 - val_loss: 1.6192
Epoch 44/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.5623 - val_loss: 1.6192
Epoch 45/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 40s 39ms/step - accuracy: 0.5298 - loss: 1.7505 - val_accuracy: 0.5641 - val_loss: 1.6081
Epoch 46/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.5641 - val_loss: 1.6081
Epoch 47/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 40s 40ms/step - accuracy: 0.5330 - loss: 1.7099 - val_accuracy: 0.5649 - val_loss: 1.6139
Epoch 48/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.5649 - val_loss: 1.6139
Epoch 49/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 38ms/step - accuracy: 0.5453 - loss: 1.6780 - val_accuracy: 0.5797 - val_loss: 1.5340
Epoch 50/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.5797 - val_loss: 1.5340
Epoch 51/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.5478 - loss: 1.6535 - val_accuracy: 0.5839 - val_loss: 1.5338
Epoch 52/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.5839 - val_loss: 1.5338

Epoch 53/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.5571 - loss: 1.6143 - val_accuracy: 0.5809 - val_loss: 1.5384
Epoch 54/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.5809 - val_loss: 1.5384
Epoch 55/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 38ms/step - accuracy: 0.5654 - loss: 1.5860 - val_accuracy: 0.5860 - val_loss: 1.5412
Epoch 56/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.5860 - val_loss: 1.5412
Epoch 57/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.5674 - loss: 1.5796 - val_accuracy: 0.5839 - val_loss: 1.5253
Epoch 58/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.5839 - val_loss: 1.5253
Epoch 59/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.5752 - loss: 1.5414 - val_accuracy: 0.5929 - val_loss: 1.4925
Epoch 60/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.5929 - val_loss: 1.4925
Epoch 61/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.5773 - loss: 1.5241 - val_accuracy: 0.6041 - val_loss: 1.4637
Epoch 62/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6041 - val_loss: 1.4637
Epoch 63/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 38ms/step - accuracy: 0.5866 - loss: 1.4970 - val_accuracy: 0.5987 - val_loss: 1.4755
Epoch 64/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.5987 - val_loss: 1.4755
Epoch 65/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.5881 - loss: 1.4767 - val_accuracy: 0.6031 - val_loss: 1.4516
Epoch 66/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6031 - val_loss: 1.4516
Epoch 67/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.6023 - loss: 1.4336 - val_accuracy: 0.6063 - val_loss: 1.4343
Epoch 68/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6063 - val_loss: 1.4343

Epoch 69/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.5995 - loss: 1.4272 - val_accuracy: 0.6047 - val_loss: 1.4483
Epoch 70/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6047 - val_loss: 1.4483
Epoch 71/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 40s 39ms/step - accuracy: 0.6048 - loss: 1.4037 - val_accuracy: 0.6025 - val_loss: 1.4570
Epoch 72/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6025 - val_loss: 1.4570
Epoch 73/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 38ms/step - accuracy: 0.6090 - loss: 1.3815 - val_accuracy: 0.6078 - val_loss: 1.4405
Epoch 74/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6078 - val_loss: 1.4405
Epoch 75/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 40s 40ms/step - accuracy: 0.6211 - loss: 1.3490 - val_accuracy: 0.6145 - val_loss: 1.4152
Epoch 76/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6145 - val_loss: 1.4152
Epoch 77/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.6208 - loss: 1.3319 - val_accuracy: 0.6135 - val_loss: 1.4037
Epoch 78/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6135 - val_loss: 1.4037
Epoch 79/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.6316 - loss: 1.2991 - val_accuracy: 0.6193 - val_loss: 1.3880
Epoch 80/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6193 - val_loss: 1.3880
Epoch 81/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 40s 39ms/step - accuracy: 0.6367 - loss: 1.2768 - val_accuracy: 0.6153 - val_loss: 1.3998
Epoch 82/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6153 - val_loss: 1.3998
Epoch 83/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.6394 - loss: 1.2687 - val_accuracy: 0.6176 - val_loss: 1.4091
Epoch 84/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6176 - val_loss: 1.4091

Epoch 85/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 39s 39ms/step - accuracy: 0.6428 - loss: 1.2562 - val_accuracy: 0.6180 - val_loss: 1.3961
Epoch 86/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6180 - val_loss: 1.3961
Epoch 87/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 38s 38ms/step - accuracy: 0.6432 - loss: 1.2382 - val_accuracy: 0.6254 - val_loss: 1.3634
Epoch 88/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6254 - val_loss: 1.3634
Epoch 89/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 37s 37ms/step - accuracy: 0.6523 - loss: 1.2095 - val_accuracy: 0.6185 - val_loss: 1.3929
Epoch 90/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6185 - val_loss: 1.3929
Epoch 91/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 38s 38ms/step - accuracy: 0.6576 - loss: 1.1940 - val_accuracy: 0.6284 - val_loss: 1.3665
Epoch 92/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6284 - val_loss: 1.3665
Epoch 93/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 38s 38ms/step - accuracy: 0.6621 - loss: 1.1833 - val_accuracy: 0.6385 - val_loss: 1.3389
Epoch 94/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6385 - val_loss: 1.3389
Epoch 95/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 37s 37ms/step - accuracy: 0.6621 - loss: 1.1579 - val_accuracy: 0.6387 - val_loss: 1.3340
Epoch 96/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6387 - val_loss: 1.3340
Epoch 97/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 38s 37ms/step - accuracy: 0.6697 - loss: 1.1463 - val_accuracy: 0.6342 - val_loss: 1.3409
Epoch 98/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6342 - val_loss: 1.3409
Epoch 99/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 37s 37ms/step - accuracy: 0.6716 - loss: 1.1252 - val_accuracy: 0.6381 - val_loss: 1.3399
Epoch 100/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6381 - val_loss: 1.3399

```
Epoch 101/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 37s 37ms/step - accuracy: 0.6771
- loss: 1.1106 - val_accuracy: 0.6396 - val_loss: 1.3241
Epoch 102/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy:
0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6396 - val_loss: 1.3241
Epoch 103/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 37s 37ms/step - accuracy: 0.6885
- loss: 1.0751 - val_accuracy: 0.6380 - val_loss: 1.3497
Epoch 104/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy:
0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6380 - val_loss: 1.3497
Epoch 105/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 37s 37ms/step - accuracy: 0.6884
- loss: 1.0709 - val_accuracy: 0.6444 - val_loss: 1.3227
Epoch 106/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy:
0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6444 - val_loss: 1.3227
Epoch 107/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 37s 37ms/step - accuracy: 0.6863
- loss: 1.0744 - val_accuracy: 0.6412 - val_loss: 1.3215
Epoch 108/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy:
0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6412 - val_loss: 1.3215
Epoch 109/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 37s 37ms/step - accuracy: 0.6954
- loss: 1.0426 - val_accuracy: 0.6373 - val_loss: 1.3374
Epoch 110/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy:
0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6373 - val_loss: 1.3374
Epoch 111/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 37s 37ms/step - accuracy: 0.6983
- loss: 1.0315 - val_accuracy: 0.6413 - val_loss: 1.3399
Epoch 112/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy:
0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6413 - val_loss: 1.3399
Epoch 113/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 37s 37ms/step - accuracy: 0.6978
- loss: 1.0335 - val_accuracy: 0.6417 - val_loss: 1.3300
Epoch 114/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy:
0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 0.6417 - val_loss: 1.3300
Epoch 115/600
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 37s 37ms/step - accuracy: 0.7018
- loss: 1.0163 - val_accuracy: 0.6387 - val_loss: 1.3867
Epoch 115: early stopping
Restoring model weights from the end of the best epoch: 107.
Test loss: 1.3215161561965942 / Test accuracy: 0.6412000060081482
```

Efficientnet - B3:

model = create_model('efficientnet_b3', pretrained=False, num_classes=num_classes)

Below are the observation found:

Epoch 81/120, Loss: 0.0008397004608532025, accuracy: 0.00%
6721.0s 88 Epoch 82/120, Loss: 0.0008494730037121291, accuracy: 0.00%
6802.5s 89 Epoch 83/120, Loss: 0.0010659214445534221, accuracy: 0.00%
6884.1s 90 Epoch 84/120, Loss: 0.0009784858276488412, accuracy: 0.00%
6965.7s 91 Epoch 85/120, Loss: 0.001248635907393668, accuracy: 0.00%
7047.2s 92 Epoch 86/120, Loss: 0.001824407042995172, accuracy: 0.00%
7128.9s 93 Epoch 87/120, Loss: 0.0021520739340568197, accuracy: 0.00%
7210.4s 94 Epoch 88/120, Loss: 0.0022604803016292863, accuracy: 0.00%
7292.1s 95 Epoch 89/120, Loss: 0.0023133259165672515, accuracy: 0.00%
7373.6s 96 Epoch 90/120, Loss: 0.04077925217703362, accuracy: 0.00%
7455.3s 97 Epoch 91/120, Loss: 0.13161427555959435, accuracy: 0.00%
7536.8s 98 Epoch 92/120, Loss: 0.13119682580043998
7618.4s 99 Epoch 93/120, Loss: 0.10303213900025887
7700.0s 100 Epoch 94/120, Loss: 0.08429403640801393
7781.6s 101 Epoch 95/120, Loss: 0.07916979428050638
7863.2s 102 Epoch 96/120, Loss: 0.07520879628254643
7944.8s 103 Epoch 97/120, Loss: 0.06565497622271127
8026.4s 104 Epoch 98/120, Loss: 0.05923773659558236
8108.0s 105 Epoch 99/120, Loss: 0.12180037037292614
8189.8s 106 Epoch 100/120, Loss: 0.11119969401367102