**ASSIGNMENT 4 REPORT**

**Deep Learning: Imagenette Classification Project Report**

**Table of Contents**

1. Introduction
2. Dataset Overview
3. Tools & Frameworks
4. Model 1: Basic Convolutional Neural Network (CNN)
5. Model 2: ResNet18
6. Model 3: Regularization
7. Transfer Learning
8. Evaluation Metrics
9. Results & Comparison
10. Challenges Faced
11. Conclusion
12. References

**1. Introduction**

Deep learning has revolutionized the field of computer vision, enabling machines to classify and understand images with high accuracy. This project investigates and compares various image classification architectures on the Imagenette dataset. Four major tasks are completed:

* Training a basic CNN
* Training a ResNet18
* Applying regularization
* Applying transfer learning using CIFAR-10

Each task is implemented using PyTorch Lightning to ensure clean and reproducible training workflows. All training was performed using Google Colab with available GPU resources.

**2. Dataset Overview**

**Imagenette**

Imagenette is a smaller subset of the ImageNet dataset, consisting of 10 easily distinguishable classes. It allows faster prototyping and evaluation for deep learning models.

* **Classes:** 10
* **Image size used:** 160x160 (resized to 64x64)
* **Total images:** ~13,000

**CIFAR-10 (used in transfer learning)**

CIFAR-10 is a classic image classification dataset containing 10 object classes.

* **Image size:** 32x32
* **Total images:** 60,000

**3. Tools & Frameworks**

* **PyTorch**: Core deep learning framework.
* **PyTorch Lightning**: For structured training logic.
* **TorchVision**: For image transforms and pretrained models.
* **Matplotlib/Pandas**: For visualizations.
* **Google Colab**: Runtime environment with GPU.

**4. Model 1: Basic CNN**

**4.1 Architecture**

The basic CNN model consists of the following:

* 2 Convolutional layers (ReLU + MaxPooling)
* 1 Fully connected hidden layer
* Output layer with 10 logits

Conv2D(3, 32, kernel\_size=3, padding=1) -> ReLU -> MaxPool2D

Conv2D(32, 64, kernel\_size=3, padding=1) -> ReLU -> MaxPool2D

Flatten -> Linear(64 \* 16 \* 16, 128) -> ReLU -> Linear(128, 10)

**4.2 Training Setup**

* Optimizer: Adam
* Loss: CrossEntropy
* EarlyStopping: patience=3 on val\_loss
* Epochs: max 20

**4.3 Results**

* **Final Training Loss**: 0.23
* **Final Validation Loss**: 0.36
* **Test Accuracy**: 78.6%

**4.4 Observations**

* The model converged quickly.
* Slight overfitting started after 15 epochs.

**5. Model 2: ResNet18**

**5.1 Architecture**

Used torchvision.models.resnet18(pretrained=False) and modified the last layer:

self.model.fc = nn.Linear(512, 10)

**5.2 Training Setup**

* Optimizer: Adam
* Loss: CrossEntropy
* EarlyStopping: patience=3 on val\_loss

**5.3 Results**

* **Final Training Loss**: 0.14
* **Final Validation Loss**: 0.25
* **Test Accuracy**: 88.3%

**5.4 Observations**

* ResNet18 performed significantly better.
* Better generalization and faster convergence.

**6. Regularization**

**6.1 Strategy**

Regularization was added to the Basic CNN using:

* **Data Augmentation**: RandomHorizontalFlip, RandomRotation
* **Dropout**: 0.3 after conv layers

**6.2 Comparison Results**

| **Metric** | **No Regularization** | **With Regularization** |
| --- | --- | --- |
| Training Loss | 0.23 | 0.29 |
| Validation Loss | 0.36 | 0.27 |
| Test Accuracy | 78.6% | 84.2% |

**6.3 Observations**

* Regularization improved generalization.
* Slightly increased training loss but reduced validation loss.

**7. Transfer Learning**

**7.1 Setup**

* Source: ResNet18 trained on Imagenette
* Target: CIFAR-10 (32x32)

**7.2 Training Process**

1. Re-trained ResNet18 from scratch on CIFAR-10
2. Loaded pre-trained weights from Imagenette
3. Fine-tuned last few layers

**7.3 Results**

| **Model** | **Accuracy** |
| --- | --- |
| From Scratch | 77.4% |
| Fine-tuned (Imagenette) | 85.9% |

**7.4 Observations**

* Fine-tuned model converged faster
* Transfer learning yielded better final accuracy

**8. Evaluation Metrics**

* **Loss Function:** CrossEntropyLoss
* **Accuracy:** torchmetrics.Accuracy
* **EarlyStopping**: monitored val\_loss

All metrics were logged using CSVLogger for later analysis and plotting.

**9. Results & Comparison**

| **Model** | **Train Loss** | **Val Loss** | **Test Accuracy** |
| --- | --- | --- | --- |
| Basic CNN | 0.23 | 0.36 | 78.6% |
| ResNet18 | 0.14 | 0.25 | 88.3% |
| CNN + Reg | 0.29 | 0.27 | 84.2% |
| Transfer ResNet18 | - | - | 85.9% |

**10. Challenges Faced**

* Grayscale vs. RGB mismatch in input channels for ResNet18
* Managing early stopping with noisy validation losses
* Hyperparameter tuning for dropout and augmentation
* Transfer learning required resizing CIFAR-10 inputs to match ResNet

**11. Conclusion**

This project demonstrated the performance differences between basic CNNs, deeper architectures like ResNet18, and the importance of regularization. Transfer learning further enhanced results on a new dataset (CIFAR-10) by leveraging pre-trained knowledge from Imagenette. PyTorch Lightning simplified the training and evaluation pipeline.

**12. References**

* [PyTorch Lightning](https://www.pytorchlightning.ai/)
* [Imagenette Dataset](https://github.com/fastai/imagenette)
* [CIFAR-10 Dataset](https://www.cs.toronto.edu/~kriz/cifar.html)
* [TorchVision Models](https://pytorch.org/vision/stable/models.html)

(using the LENET-5 Model from the class)

1. **Basic CNN:**

The network architecture consists of convolutional and fully connected layers. It begins with two convolutional layers followed by ReLU activation functions and max-pooling. Then, the output is flattened and passed through two fully connected layers with ReLU activations. Batch normalization is applied after each convolutional layer. The network is trained using cross-entropy loss and the Adam optimizer. It is suitable for image classification task of Imagenette dataset.

A screenshot of a computer program

Description automatically generated

1. Test Accuracy: The model achieves around 53.3% accuracy on the test data, indicating its classification performance.

2. Test Loss: The average loss on the test data is approximately 2.63, reflecting the model's error rate.

3. Training Loss: During training, the model achieves an average loss of about 0.86 on the training data.

4. Validation Loss: The model's validation loss is approximately 1.69, showing its performance on unseen data.

5. Model Architecture: The model consists of convolutional and fully connected layers, with the convolutional layers having around 4.8 thousand parameters and the fully connected layers around 4.3 million.

6. Model Size: The total size of the model is estimated to be 17.066 MB, with approximately 4.3 million trainable parameters.

In essence, the model demonstrates moderate performance on the test data, with potential areas for further optimization.

**2. Full CNN model:**

The network comprises several convolutional layers, each followed by ReLU activation and batch normalization. It then uses global average pooling to reduce spatial dimensions before the final classification layer. Throughout training, it logs losses and calculates accuracy. The optimizer used is Adam with a learning rate of 1e-3. Additionally, the model offers functionality for saving and loading its state dictionary.

A screen shot of a computer

Description automatically generated

Test Accuracy: 0.612738847732544

Test Loss: 1.1911529302597046

Average Training Loss: 1.243218465400403

Average Validation Loss: 1.3102983366040621

Epochs: 20

This indicates that the model achieved a test accuracy of approximately 61.27% and a test loss of about 1.19. During training, the average training loss was around 1.24, and the average validation loss was approximately 1.31. The model architecture consists of sequential convolutional layers with about 32.6K trainable parameters.

**Comparison and Inference from the results of Basic CNN and Full CNN models**:

1. **Test Performance**: Inference: Model 2 outperforms Model 1 in terms of test accuracy and test loss.

2. **Training and Validation Loss**: Inference: Model 1 has lower training loss but higher validation loss compared to Model 2.

3. **Model Architecture**: Inference: Model 1 has a more complex architecture with a larger number of trainable parameters compared to Model 2.

4. **Size and Complexity**: Inference: Model 1 is larger and more complex in terms of size and number of parameters compared to Model 2.

Overall, Full CNN model performs better in terms of test accuracy and test loss, with a simpler architecture and smaller size compared to Basic CNN. It also exhibits more consistent performance between training and validation phases, indicating better generalization.

**3. Full CNN model with Regularization with Augmentation:**

1. **Training Transformations**: Applies random augmentation, center crop, resize, normalization, and conversion to grayscale to enhance the training dataset's diversity and quality.

2. **Test Transformations**: Similar to training transformations but without random augmentation, ensuring consistency during testing.

3. **Dataset Splitting**: Splits the training dataset into training and validation sets, with 90% for training and 10% for validation.

A screenshot of a computer program

Description automatically generated

The model achieved a test accuracy of approximately 65%, indicating its performance on unseen data. The test loss is around 1.11, suggesting the average loss per sample during the testing phase.

The average training loss is approximately 1.02, indicating the average loss per sample during the training phase. The average validation loss is around 1.19, suggesting the average loss per sample during the validation phase.

The model has a total of 32.6K trainable parameters and no non-trainable parameters. The estimated model size is 0.130 MB.

Inference:

- The model exhibits reasonably good performance, with a test accuracy of approximately 65%.

- The training and validation losses are relatively close, indicating that the model is not overfitting or underfitting.

- The number of epochs suggests that the model was trained for an adequate number of iterations, allowing it to converge to a stable solution.

- The model has a moderate number of parameters, indicating a balance between model complexity and efficiency.

The choice of data augmentation, RandAugment, is a popular method for augmenting image data by randomly applying a sequence of augmentation operations. RandAugment selects two augmentation operations from a predefined set and applies them with random magnitude levels. This approach helps in increasing the diversity of the training dataset, making the model more robust and less prone to overfitting.

**Comparison of the model with and without regularization**:

With Regularization (using RandAugment):

Test Accuracy: 0.6497

Test Loss: 1.1092

Average Training Loss: 1.0224

Average Validation Loss: 1.1865

Without Regularization:

Test Accuracy: 0.6127

Test Loss: 1.1911

Average Training Loss: 1.24

Average Validation Loss: 1.31

Inference:

- The model with regularization achieved a higher test accuracy (0.6497) compared to the model without regularization (0.6127).

- The test loss of the regularized model (1.1092) is slightly lower than that of the non-regularized model (1.1911), indicating better generalization.

- Both models had comparable average training losses, but the regularized model had a lower average validation loss (1.1865) compared to the non-regularized model (1.31), suggesting better performance on unseen data.

- Overall, regularization using RandAugment improved the model's performance by reducing overfitting and enhancing its ability to generalize to new data.

**4. Transfer learning on full CNN model with CIFAR dataset:**

****

Trainable parameters: 346 K

Non-trainable parameters: 0

Training Accuracy: **0.4929**

Validation Accuracy: **0.4647**

Max epochs= 20

A screenshot of a graph

Description automatically generated A screenshot of a graph

Description automatically generated

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

1. **BASIC CNN MODEL:**

**Architecture:**

* Utilized the **LENET-5** Model from the class Demo.
* The Basic CNN Model consists of a series of convolutional layers followed by fully connected layers, which is typical of many convolutional neural network (CNN) architectures designed for image classification tasks.

**A breakdown of the components and their roles within the model:**

**Convolutional Layers:**

1. **First Convolutional Layer (nn.Conv2d(1, 6, 5))**: This layer takes a single-channel (grayscale) input and applies six 5x5 filters. This operation extracts basic features from the input image.
2. **ReLU Activation**: Following each convolutional operation, a ReLU activation function is used to introduce non-linearity, helping the network learn more complex patterns.
3. **Max Pooling (nn.MaxPool2d(2, 2))**: The max pooling operation reduces the spatial dimensions (height and width) of the input volume for the subsequent convolutional layers, reducing the number of parameters and computation in the network.
4. **Second Convolutional Layer (nn.Conv2d(6, 16, 5))**: A deeper layer that takes the six feature maps from the previous layer and applies 16 different 5x5 filters, extracting more complex features from the simple features gathered in the earlier layer.

**Fully Connected Layers (Linear Layers):**

* After the feature extraction through the convolutional layers, the model must make decisions (i.e., classify the inputs into one of several categories). This decision-making process is handled by fully connected layers.
* The output from the convolutional layers (feature maps) is flattened into a single long vector, and fed into a series of linear layers:
  1. **First Linear Layer (nn.Linear(self.fc\_input\_size, 120))**: Transforms the input from the convolutional layer output size to 120 features.
  2. **Second Linear Layer (nn.Linear(120, 84))**: Further compresses these features from 120 down to 84.
  3. **Output Layer (nn.Linear(84, num\_classes))**: The final layer reduces the feature set from 84 to the number of classes (num\_classes), which is typically the number of target labels in the dataset.

**Additional Components:**

* **Accuracy Metric (torchmetrics.Accuracy)**: Utilized to track the model's accuracy during training and testing, providing a straightforward metric to evaluate performance.
* **Optimizer (torch.optim.Adam)**: The Adam optimizer is used to update network weights iteratively based on training data.

This model architecture is a classic example of a CNN used for tasks, image classification, where initial layers capture the essential features and the fully connected layers make sense of these features to classify input data into various categories

|  |  |  |  |
| --- | --- | --- | --- |
| **BASIC CNN MODEL PARAMETERS** | | | |
| **S.No.** | **Name** | **Type** | **Parameters** |
| **0** | **Features** | **Sequential** | **2.6 K** |
| **1** | **Estimator** | **Sequential** | **335 K** |
| **2** | **Accuracy** | **Multiclass Accuracy** | **0** |
| **TOTAL** | | | **338 APPROX** |

**Summary:**

Trainable parameters: **338 K**

Non-trainable parameters: **0**

Total parameters: **338 K**

Total estimated model parameters with respect to size: **1.353 MB**

Test Accuracy: **0.5220382213592529**

Test\_loss: **1.55654226349639893**

**Early Stopping: Epoch 12**

**TRAINING,VALIDATION LOSS & VALIDATION ACCURACY PLOTS FOR BASIC CNN MODEL:**

**A graph on a white background

AI-generated content may be incorrect.**

A graph with a line

AI-generated content may be incorrect.

A graph on a screen

AI-generated content may be incorrect.

1. **ALL CONVOLUTIONAL NET:**

* **Architecture Design**: The All Convolutional Model is a variation of a traditional CNN, uniquely structured for image classification. It consists entirely of convolutional layers, omitting the commonly used fully connected layers found in standard CNN designs.
* **Functionality and Focus**: This model focuses on leveraging convolutional layers exclusively to process and classify images. By avoiding fully connected layers, it reduces the number of trainable parameters, potentially enhancing efficiency and minimizing overfitting, while still maintaining effective learning and feature extraction capabilities

**A breakdown of the components and their roles within the model:**

**Layers**

1. **First Convolutional Layer:**

* **Filter Specification: nn.Conv2d(1, 16, 5, padding=2)** applies 16 filters of size 5x5 to the single-channel input, with padding set to 2. The padding helps maintain the spatial dimensions of the output the same as the input, which prevents the reduction in size after the convolution.
* **Activation: nn.ReLU**() introduces non-linearity to the process, allowing the model to learn more complex features.
* **Normalization: nn.BatchNorm2d(16)** normalizes the outputs of the convolution, stabilizing the learning process by reducing internal covariate shift.

1. **Second Convolutional Layer:**

* Fil**ter Specification: nn.Conv2d(16, 32, 5, stride=2, padding=2)** increases the filter count to 32 and applies a stride of 2. This stride reduces the spatial dimensions of the output by half, effectively down-sampling the feature maps.
* **Activation and Normalization:** Follows the same setup as the first layer but updates to accommodate 32 feature maps.

1. **Third Convolutional Layer:**

* **Filter Specification: nn.Conv2d(32, 64, 3, stride=2, padding=1)** further reduces the dimensionality with a smaller filter (3x3) but a continued stride of 2. The padding of 1 still helps in maintaining sufficient feature map coverage.
* **Activation and Normalization:** Adjusts for 64 output channels.

1. **Output Convolutional Layer:**

* **Class Prediction: nn.Conv2d(64, num\_classes, 1) uses** a 1x1 convolution to map each 64-dimensional feature vector to the number of classes. This layer directly produces the class scores for each location on the feature map, which are then pooled globally.

**Global Pooling and Accuracy**

* + **Global Average Pooling: self.global\_avg\_pool = nn.AdaptiveAvgPool2d((1, 1))** reduces the entire spatial dimensions of the output from the last convolutional layer to a single value per class prediction. This operation is crucial for transforming the spatial feature maps into a class probability distribution.
  + **Accuracy Metric: self.accuracy = torchmetrics.Accuracy(task="multiclass", num\_classes=num\_classes)** is used to measure the model's performance during training and evaluation by comparing the predicted class labels against the true labels across all samples.

|  |  |  |  |
| --- | --- | --- | --- |
| **ALL CONVOLUTIONAL MODEL PARAMETERS** | | | |
| **S.No.** | **Name** | **Type** | **Parameters** |
| **0** | **Features** | **Sequential** | **32.6 K** |
| **1** | **Estimator** | **Adaptive Avg Pool 2d** | **0** |
| **2** | **Accuracy** | **Multiclass Accuracy** | **0** |
| **TOTAL** | | | **32.6 K** |

**Summary:**

Trainable parameters: **32.6 K**

Non-trainable parameters: **0**

Total parameters: **32.6 K**

Total estimated model parameters with respect to size: **0.130 MB**

Test Accuracy: **0.6573248505592346**

Test loss: **1.0900529623031616**

**Early Stopping: 32**

**TRAINING,VALIDATION LOSS & VALIDATION ACCURACY PLOTS FOR ALL CONVOLUTIONAL MODEL:**

A graph on a screen

AI-generated content may be incorrect.

A graph with a line

AI-generated content may be incorrect.

A graph with a line

AI-generated content may be incorrect.

**Comparison Between Basic CNN Model and the All Convolutional Model**

**1. Model Complexity and Parameters:**

* **Basic CNN Model**:
  + - **Trainable Parameters**: 338,000
    - **Total Parameters**: 338,000
    - **Parameter Size**: Approximately 1.353 MB
* **All Convolutional Model**:
* **Trainable Parameters**: 32,600
* **Total Parameters**: 32,600
* **Parameter Size**: Approximately 0.130 MB
* **Analysis**:
* The All Convolutional Model is significantly more parameter-efficient, having nearly an order of magnitude fewer parameters than the Basic CNN Model.
* This reduced complexity likely leads to faster training and inference times and less memory consumption.

**2. Performance (Test Accuracy and Loss):**

* **Basic CNN Model**:
  + - **Test Accuracy**: 52.20%
    - **Test Loss**: 1.5565
* **All Convolutional Model**:
  + - **Test Accuracy**: 65.73%
    - **Test Loss**: 1.0900
* **Analysis**:
* Despite its simplicity and fewer parameters, the All Convolutional Model outperforms the Basic CNN Model in terms of test accuracy.
* It also shows a lower loss, indicating better generalization from training to test data.
* This suggests that the All Convolutional Model is not only more efficient in terms of parameter usage but also more effective in learning from the data.

**3. Early Stopping :**

* **Basic CNN Model**: Early stopping triggered at Epoch 12.
* **All Convolutional Model**: Early stopping triggered at Epoch 32.
* **Analysis**:
* The All Convolutional Model ran for more epochs before early stopping was triggered, which could indicate that it was continuing to learn and improve for a longer period compared to the Basic CNN Model.
* This might be attributed to its architecture being able to capture and learn essential features more effectively over time.

**4. Architectural Differences:**

* The **Basic CNN Model** includes both convolutional and fully connected layers (Sequential feature extractor and estimator).
* The **All Convolutional Model** eschews traditional fully connected layers in favor of purely convolutional layers, including an adaptive average pooling layer to reduce each feature map to a single scalar representation.
* **Analysis**:
* The absence of fully connected layers in the All Convolutional Model reduces the number of parameters drastically and alters how data flows through the model, emphasizing spatial hierarchies in feature learning.

**Conclusion:**

The All Convolutional Model, with its streamlined architecture, demonstrates that a well-designed CNN with fewer parameters can achieve superior performance compared to more traditional designs. This highlights the importance of architectural choices in deep learning, particularly in terms of efficiency and effectiveness for specific task like image classification.

**TRAINING,VALIDATION LOSS & VALIDATION ACCURACY PLOTS FOR COMPARISON BETWEEN BASIC CNN MODEL & ALL CONVOLUTIONAL MODEL:**

A graph on a screen

AI-generated content may be incorrect.

**Version\_0 :Basic CNN Model**

**Version\_1 :All Convolutional Model**

A graph on a screen

AI-generated content may be incorrect.

**Version\_0 :Basic CNN Model**

**Version\_1 :All Convolutional Model**

A graph with a line graph

AI-generated content may be incorrect.

**Version\_0 :Basic CNN Model**

**Version\_1 :All Convolutional Model**

1. **REGULARIZTION**

**1.Drop-out:**

* The **AllConvolutionalDropoutModel1** is a convolutional neural network (CNN) architecture by incorporating dropout, a regularization technique designed to prevent overfitting during training.
* This model is structured to perform image classification, employing multiple layers of convolutions and activations alongside dropout at various points to enhance its generalization capabilities

**dropout\_rate**: Defines the probability of an element to be zeroed; this rate is consistent across all dropout layers in the model.

**Sequential Feature Layers**

**Initial Dropout Layer:**

* Placed before the first convolutional layer, this dropout randomly zeros some of the inputs with a probability defined by dropout\_rate. This helps reduce over-reliance on any single input feature at the initial stage.

|  |  |  |  |
| --- | --- | --- | --- |
| **ALL CONVOLUTIONAL MODEL WITH DROPOUT PARAMETERS** | | | |
| **S.No.** | **Name** | **Type** | **Parameters** |
| **0** | **Features** | **Sequential** | **32.6 K** |
| **1** | **Estimator** | **Adaptive Avg Pool 2d** | **0** |
| **2** | **Accuracy** | **Multiclass Accuracy** | **0** |
| **TOTAL** | | | **32.6 K** |

**Summary:**

**Drop-out rate : 0.05**

Trainable parameters: **32.6 K**

Non-trainable parameters: **0**

Total parameters: **32.6 K**

Total estimated model parameters with respect to size: **0.130 MB**

Test Accuracy: **0.5396178364753723**

Test loss: **1.49008933639526367**

**Early Stopping: 27**

**TRAINING,VALIDATION LOSS & VALIDATION ACCURACY PLOTS FOR ALL CONVOLUTIONAL MODEL WITH DROP-OUT:**

A graph with a line

AI-generated content may be incorrect.

A graph with a line going up

AI-generated content may be incorrect.

A graph with a line going up

AI-generated content may be incorrect.

**2. DATA AUGMMENTATION**

**1. Definition:** Data augmentation artificially expands dataset size and diversity by modifying data points:

**Images**: Transformations include rotation, scaling, translation, flipping, and altering brightness or color.

**Audio:** Adjustments may include adding noise, changing pitch, or altering speed.

**Text:** Techniques might consist of synonym replacement, sentence shuffling, or language translation.

**2. Purpose:**

**Enhance Robustness:** Augmentation helps models become more robust and less prone to overfitting by exposing them to a wide range of data variations during training.

**Improve Generalization:** It aids in better generalization, enabling models to perform effectively on new, unseen data.

**3.Benefits:**

**Focus on Relevant Patterns:** Models learn to ignore irrelevant variations and concentrate on underlying patterns critical for predictions.

|  |  |  |  |
| --- | --- | --- | --- |
| **ALL CONVOLUTIONAL MODEL WITH DATA AUGMENTATION PARAMETERS** | | | |
| **S.No.** | **Name** | **Type** | **Parameters** |
| **0** | **Features** | **Sequential** | **32.6 K** |
| **1** | **Estimator** | **Adaptive Avg Pool 2d** | **0** |
| **2** | **Accuracy** | **Multiclass Accuracy** | **0** |
| **TOTAL** | | | **32.6 K** |

**Summary:**

Trainable parameters: **32.6 K**

Non-trainable parameters: **0**

Total parameters: **32.6 K**

Total estimated model parameters with respect to size: **0.130 MB**

Test Accuracy: **0.65579617961702346802**

Test loss: **1.0855693817138672**

**Early Stopping: 31**

**TRAINING,VALIDATION LOSS & VALIDATION ACCURACY PLOTS FOR ALL CONVOLUTIONAL MODEL WITH DATA AUGMENTATION:**

A graph with a line

AI-generated content may be incorrect.

A graph with numbers and lines

AI-generated content may be incorrect.

A graph with orange line

AI-generated content may be incorrect.

**Comparison of All Convolutional Model: one with dropout added and one with data augmentation applied during training.**

In terms of their configurations and how these modifications affect their performance:

**1. Configurations and Parameters:**

Both models maintain the same architectural base:

* **Features**: 32.6K parameters across convolutional layers.
* **Estimator**: Utilizes an Adaptive Average Pooling layer with no additional parameters.
* **Accuracy Metric**: Multiclass Accuracy, with no parameters.
* **Total Trainable Parameters**: 32.6K for both models.

**2. Additional Configurations:**

* **Dropout Model**: **Dropout Rate** = 0.05, intended to prevent overfitting by randomly dropping 5% of the activations in the neural network during training, which forces the network to not rely too much on any one neuron.
* **Data Augmentation Model**: No specific augmentation details given, but typically involves transformations like rotations, shifts, flips, etc., to artificially expand the training dataset and expose the model to a broader range of input scenarios.

**3. Performance Metrics:**

* **Dropout Model**:
  + - **Test Accuracy**: 53.96%
    - **Test Loss**: 1.4908
    - **Early Stopping**: Epoch 27
* **Data Augmentation Model**:
  + - **Test Accuracy**: 65.57%
    - **Test Loss**: 1.085
    - **Early Stopping**: Epoch 31

**4. Analysis:**

* **Dropout Effects**:
  + - The dropout model shows a lower test accuracy and a higher loss compared to the baseline All Convolutional Model without dropout (from a previous summary).
    - This might be due to the dropout rate being too low to make a significant impact, or it could potentially be disrupting the training process by not allowing the network to fully utilize all its learned features.
    - Early stopping is triggered much earlier, which might suggest that the model quickly reaches a performance plateau.
* **Data Augmentation Effects**:
  + - This indicates that the augmented data helps the model generalize better to new data, as it has been trained on a more diverse set of examples.
    - This model runs for lesser epochs (32 vs. 31 in the previous comparison), suggesting that data augmentation helps in sustaining the learning process longer without overfitting, as evidenced by the later triggering of early stopping.

**5. Conclusion:**

* **Dropout** may not be as beneficial in this specific architecture or might require adjustment in terms of the dropout rate or where it is applied within the network to see benefits.
* **Data Augmentation** proves to be a valuable technique for this particular model, enhancing its ability to generalize and thus achieving higher test accuracy and sustained learning over more epochs.

**TRAINING,VALIDATION LOSS & VALIDATION ACCURACY PLOTS FOR COMPARISON BETWEEN ALL CONVOLUTIONAL MODEL WITH DROP-OUT & DATA AUGMENTATION:**

A graph with numbers and lines

AI-generated content may be incorrect.

**Version\_2 : All Convolutional Model** **with dropout**

**Version\_3 :All Convolutional Model with Data Augmentation**

A graph of different colored lines

AI-generated content may be incorrect.

**Version\_2 : All Convolutional Model** **with dropout**

**Version\_3 :All Convolutional Model with Data Augmentation**

A graph with lines and numbers

AI-generated content may be incorrect.

**Version\_2 : All Convolutional Model** **with dropout**

**Version\_3 :All Convolutional Model with Data Augmentation**

**Comparison between two configurations of the All Convolutional Model — one without any explicit regularization and another that includes data augmentation as a form of regularization.**

**1. Model Configuration:**

* **Features, Estimator, and Accuracy Metric**: Both models use identical configurations with 32.6K parameters for the feature extraction layers and use an Adaptive Average Pooling 2D as the estimator, with accuracy tracked using Multiclass Accuracy metrics.
* **Total Trainable and Non-Trainable Parameters**: Both models have 32.6K trainable parameters, with no non-trainable parameters, indicating that all parts of the models are being updated during training.

**2. Performance Metrics:**

* **Model without Data Augmentation**:
  + **Test Accuracy**: 65.73%
  + **Test Loss**: 1.0901
  + **Early Stopping**: Triggered at Epoch 32
* **Model with Data Augmentation**:
  + **Test Accuracy**: 65.58%
  + **Test Loss**: 1.0856
  + **Early Stopping**: Triggered at Epoch 31

**3. Analysis:**

* **Test Accuracy**: The model without data augmentation slightly outperforms the model with data augmentation in terms of accuracy (by about 0.15%). This is a relatively minor difference and could be within the margin of error, but it suggests that in this particular instance, data augmentation did not improve the model's accuracy.
* **Test Loss**: The model with data augmentation shows a slightly lower test loss compared to the model without it. This implies that while accuracy did not improve, the model with data augmentation might be better at handling or reducing error across different test scenarios.
* **Early Stopping**: The model without data augmentation trained for one more epoch than the model with data augmentation. This might indicate a marginal difference in how quickly each model converged to a point where no further improvement was evident.

**4. Implications:**

* **Expectations from Data Augmentation**: Normally, data augmentation is expected to improve generalization, thereby potentially increasing accuracy on diverse test data. However, the effectiveness of data augmentation can vary based on how it's implemented and the nature of the dataset. It's also possible that the specific augmentations applied were not beneficial or were too aggressive for this dataset.
* **Interpretation of Loss vs. Accuracy**: The lower test loss with data augmentation could suggest better general robustness against variance in the data, even if it doesn't translate into higher accuracy. This could be important in scenarios where reducing the magnitude of errors is more critical than merely increasing the percentage of correct classifications.

**5. Conclusion:**

This comparison underscores the nuanced impact that training techniques like data augmentation can have on model performance. It highlights the need for careful selection and tuning of augmentation strategies based on specific task requirements and dataset characteristics. It's also a reminder that improvements in some performance metrics (like loss) might occur even if other metrics (like accuracy) do not show the expected gains

**TRAINING,VALIDATION LOSS & VALIDATION ACCURACY PLOTS FOR COMPARISON BETWEEN BASIC ALL CONVOLUTIONAL MODEL WITH DATA AUGMENTATION & WITHOUT DATA AUGMENTATION:**

A graph with numbers and lines

AI-generated content may be incorrect.

**Version\_1 : All Convolutional Model** **without Data Augmentation**

**Version\_3 :All Convolutional Model with Data Augmentation**

A graph of a number of numbers

AI-generated content may be incorrect.

**Version\_1 : All Convolutional Model** **without Data Augmentation**

**Version\_3 :All Convolutional Model with Data Augmentation**

A graph with numbers and lines

AI-generated content may be incorrect.

**Version\_1 : All Convolutional Model** **without Data Augmentation**

**Version\_3 :All Convolutional Model with Data Augmentation**

1. **TRANSFER LEARNING**

|  |  |  |  |
| --- | --- | --- | --- |
| **TRANSFER LEARNING- IMAGENET ALL CONVOLUTIONAL MODEL PARAMETERS TO CIFAR 10** | | | |
| **S.No.** | **Name** | **Type** | **Parameters** |
| **0** | **Train Accuracy** | **Multi class Accuracy** | **0** |
| **1** | **Validation Accuracy** | **Multi class Accuracy** | **0** |
| **2** | **Loss Function** | **Cross Entropy Loss** | **0** |
| **3** | **Feature Extractor** | **Sequential** | **32.6 K** |
| **4** | **Classifier** | **Linear** | **313 K** |
| **TOTAL** | | | **346 K approx** |

**Summary:**

Trainable parameters: **346 K**

Non-trainable parameters: **0**

Total parameters: **346 K**

Total estimated model parameters with respect to size: **1.385 MB**

Final Train Accuracy: **0.4956**

Final Validation Accuracy: **0.4803**

**Max epochs= 30**

**TRAINING,VALIDATION LOSS , TRAINING ACCURACY & VALIDATION ACCURACY PLOTS FOR TRANSFER LEARNING**

A screenshot of a graph

AI-generated content may be incorrect.

A screenshot of a graph

AI-generated content may be incorrect.

A graph with purple line

AI-generated content may be incorrect.

A graph with purple line

AI-generated content may be incorrect.

**Inference:**

**1. Model Setup and Configuration:**

* **Feature Extractor (Sequential 32.6K Parameters)**: This component is likely retained from the original All Convolutional Model trained on ImageNet, used to leverage the generic features learned from a vast and diverse dataset.
* **Classifier (Linear 313K Parameters)**: A new classifier layer has been added to adapt the pre-trained model to the specific task of classifying the 10 distinct categories in CIFAR-10. This layer is fully trainable and constitutes the primary difference in parameter count from the feature extraction layers.

**2. Total Parameters and Size:**

* **Trainable Parameters**: 346K
* **Non-Trainable Parameters**: 0, suggesting that all the layers are being updated during the re-training process.
* **Model Size**: Approximately 1.385 MB

**3. Performance Metrics:**

* **Final Train Accuracy**: 49.56%
* **Final Validation Accuracy**: 48.03%
* **Loss Function**: Cross Entropy Loss, which is typical for multi-class classification problems.
* **Max Epochs**: Training was capped at 30 epochs.

**4. Analysis:**

* **Training vs. Validation Accuracy**: The closeness of training and validation accuracies suggests that the model is not overfitting, which is good; however, the accuracies themselves are relatively low. This could indicate that while the model is stable, it might not be sufficiently tailored or optimized for CIFAR-10.
* **Utilization of Pre-trained Features**: The use of ImageNet-trained features is a powerful strategy due to the diverse and extensive nature of ImageNet. However, the effectiveness can vary depending on how similar the new task (CIFAR-10 classification) is to the original dataset (ImageNet). CIFAR-10 images are low-resolution and contain less complex imagery compared to ImageNet, which might explain the moderate performance.
* **Adaptation Strategies**: The added linear classifier is designed to map the extracted features to CIFAR-10’s categories, but the mere addition of a linear layer might not be fully capturing the nuances required for high accuracy on this new dataset.