FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ENCS(5343) Computer Vision

Arabic Handwritten Character Recognition Using CNNs: Course Project Report

Prepared by:

Rama Abdalrahman 1191344

Laith Swies 1190513

Supervised by :

Dr. Aziz Qaroush

BIRZEIT

January – 2024

Contents

Abstract:

This project addresses the challenges and techniques associated with Arabic Handwritten Character Recognition (AHCR) using (CNNs). The primary objectives encompass the creation of a basic CNN for AHCR and exploration of advanced techniques. The assignment is structured into four tasks, each focusing on different aspects of CNN-based AHCR.

1. Introduction

CNN model

A Convolutional Neural Network, or CNN for short, is a type of neural network designed for working with grid-like data, like images. In a digital image, information is represented in a grid of pixels. Each pixel has values indicating its brightness and color. CNNs are really good at understanding and processing this kind of visual data.

CNN Layers:

CNN has three layers: a convolutional layer, a pooling layer, and a fully connected layer.

1. Convolutional Layer:

This layer is responsible for capturing features from the input data, typically an image. It applies filters (also called kernels) to small portions of the input, sliding these filters across the entire image. The convolutional operation involves element-wise multiplication of the filter values with the input data, followed by summation. This process helps the network learn patterns and features, such as edges or textures.

Output: The result is a feature map that highlights important features detected in the input.

2. Pooling Layer:

After convolution, the spatial dimensions of the feature maps are often reduced to control the computational complexity and enhance the network's ability to generalize. Pooling involves downsampling the feature maps by selecting the maximum (max pooling) or average (average pooling) values from a group of neighboring pixels. This helps retain essential information while reducing the spatial resolution.

Output: The pooled feature maps represent a condensed version of the input, focusing on the most relevant features.

3. Fully Connected Layer:

This layer is responsible for combining the features extracted by the convolutional and pooling layers to make predictions or classifications. Neurons in a fully connected layer are connected to all neurons in the previous layer, forming a dense network. The weights associated with these connections are adjusted during training to learn the relationships between features and classes.

Output: The output of the fully connected layer represents the final prediction or classification based on the learned features.

Applications of Convolutional Neural Networks (CNNs):

Object Detection:

CNNs like R-CNN, Fast R-CNN, and Faster R-CNN are crucial for object detection in areas like autonomous vehicles and facial recognition.

Semantic Segmentation:

CNN-based models, such as Deep Parsing Network and fully convolutional networks, enhance image segmentation by incorporating rich information, improving state-of-the-art semantic segmentation.

Image Captioning:

CNNs, combined with recurrent neural networks, are employed for generating captions for images and videos. This technology aids in activity recognition and describing visual content, benefiting applications like YouTube for understanding and organizing vast video content.

2. Dataset

```
Program aborted
AHDRC dataset/
    csvTrainImages 13440x1024.csv
    csvTestLabel 3360x1.csv
    csvTestImages 3360x1024.csv
    csvTrainLabel 13440x1.csv
    Test Images 3360x32x32/
        test/
            id_1226_label_25.png
            id_2050_label_17.png
            id_285_label_3.png
            id_2951_label_20.png
            id_1505_label_25.png
            id_1349_label_3.png
            id_1165_label_23.png
            id_159_label_24.png
            id_2042_label_13.png
```

```
train_df.head()
```

	id	class	path
0	12932	21	AHDRC dataset/Train Images 13440x32x32/train/id_12932_label_21.png
1	7157	27	AHDRC dataset/Train Images 13440x32x32/train/id_7157_label_27.png
2	3499	18	AHDRC dataset/Train Images 13440x32x32/train/id_3499_label_18.png
3	1520	22	AHDRC dataset/Train Images 13440x32x32/train/id_1520_label_22.png
4	5190	5	AHDRC dataset/Train Images 13440x32x32/train/id_5190_label_5.png

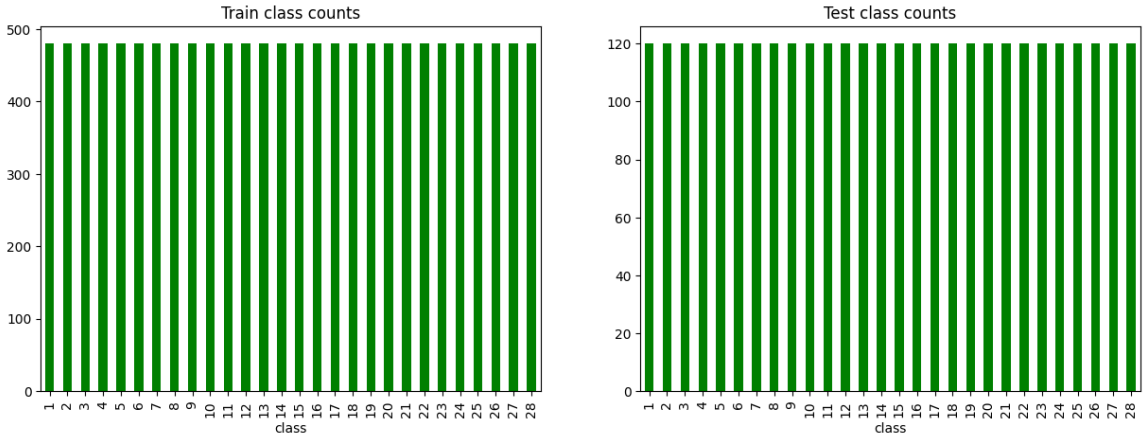
```
test_df.head()
```

	id	class	path
0	492	22	AHDRC dataset/Test Images 3360x32x32/test/id_492_label_22.png
1	2137	5	AHDRC dataset/Test Images 3360x32x32/test/id_2137_label_5.png
2	957	3	AHDRC dataset/Test Images 3360x32x32/test/id_957_label_3.png
3	2824	12	AHDRC dataset/Test Images 3360x32x32/test/id_2824_label_12.png
4	2262	11	AHDRC dataset/Test Images 3360x32x32/test/id_2262_label_11.png

```
Same unique labels in Train and Test: True
Unique labels count is 28: True
Train set length is 13440: True
Test set length is 3360: True
```

- **Explore the data and visualize it.**

Checking the distribution of the classes



All the classes are already balanced.

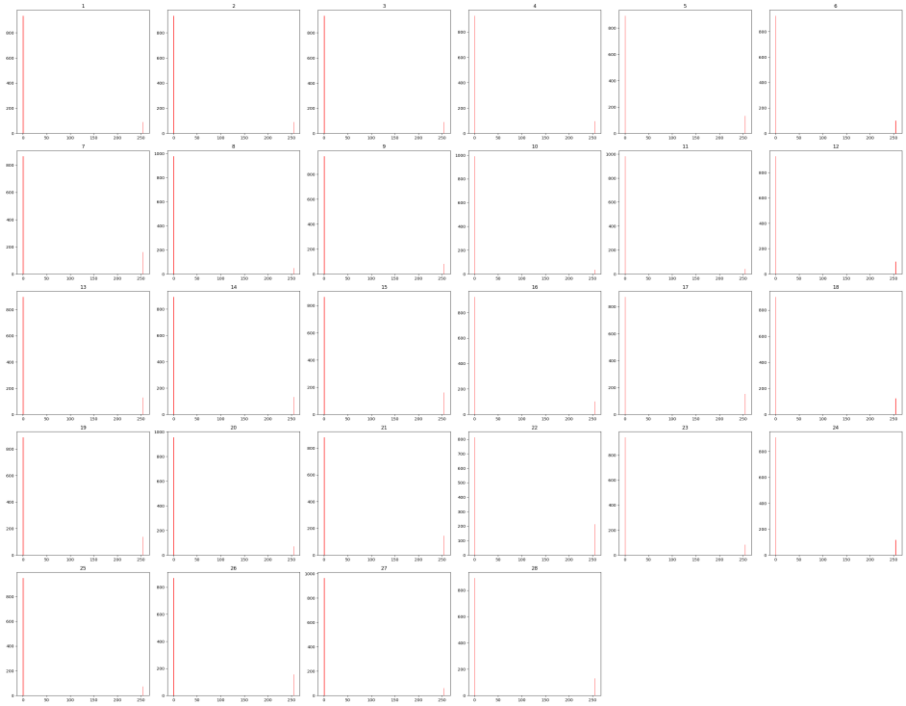
- **Missing values**

```
TRAIN SET: Number of rows containing empty values: 0
TEST SET: Number of rows containing empty values: 0
```

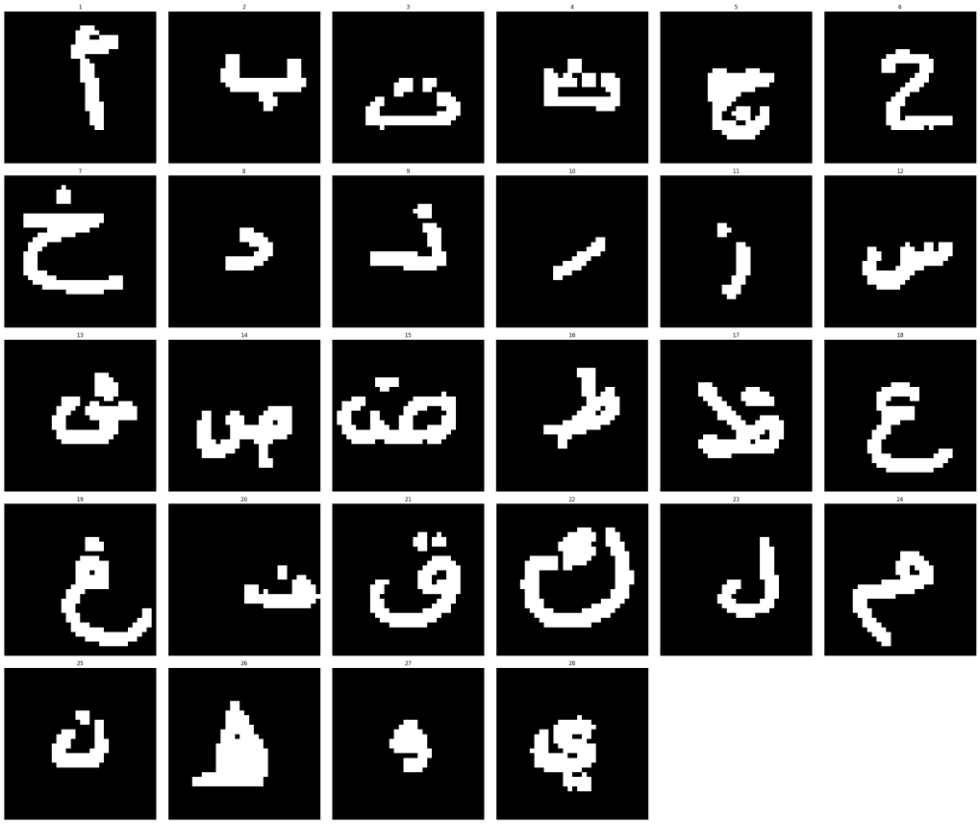
There are no rows containing empty values.

- **Sample image of each class and its histogram**

Data is binarized **,** either 0 , or 255 , as shown in images below which taken from dataset , and as shown in histograms for classes**.**



- **Printing a sample of each class**



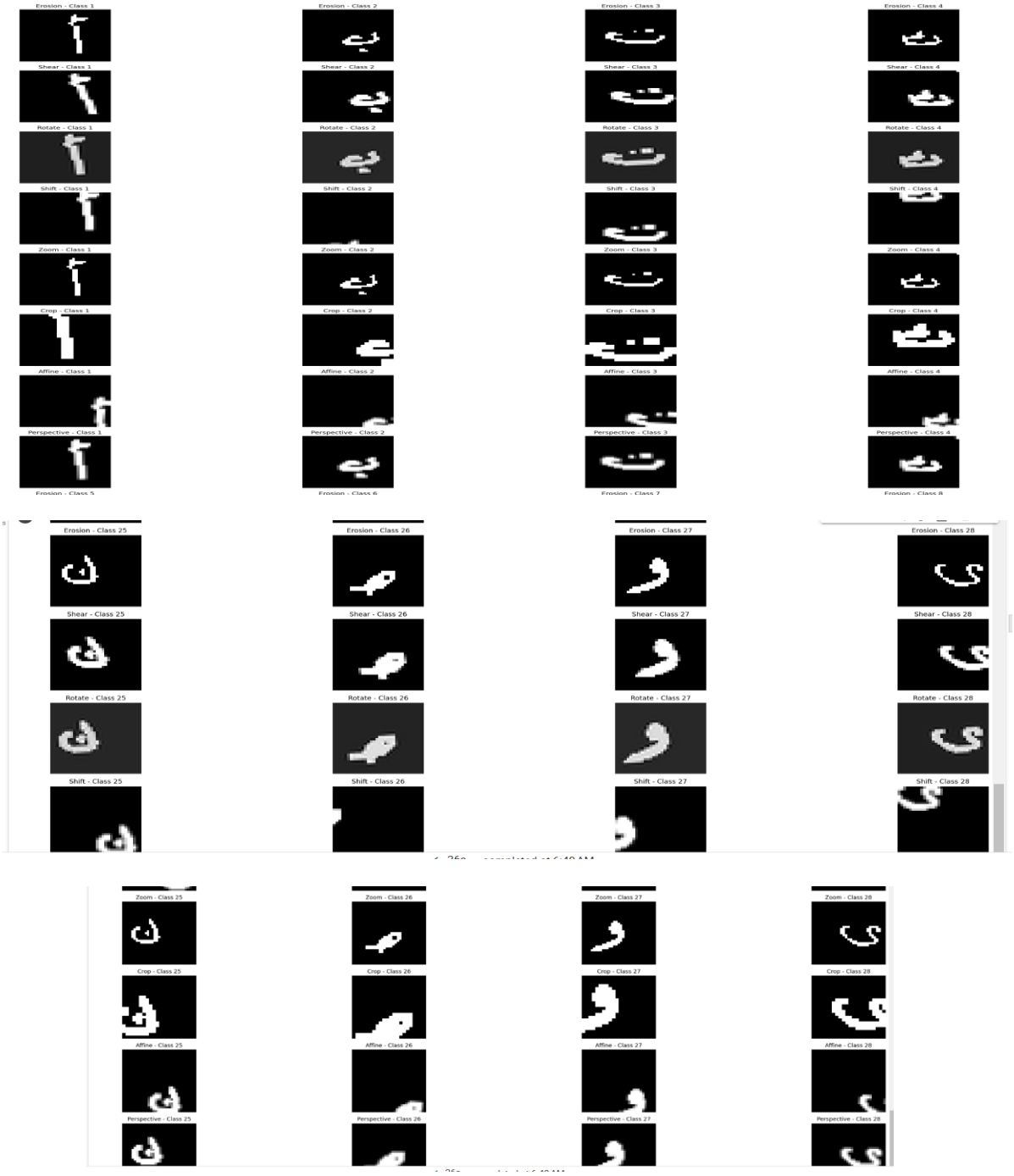
From the histograms above, the images appear to already have been binarized (pixel value can take either 0 or 255). The range will be adjusted when constructing the preprocessing method to (0 - 1).

- ## Data Augmentation Methods Visualized on Our Data

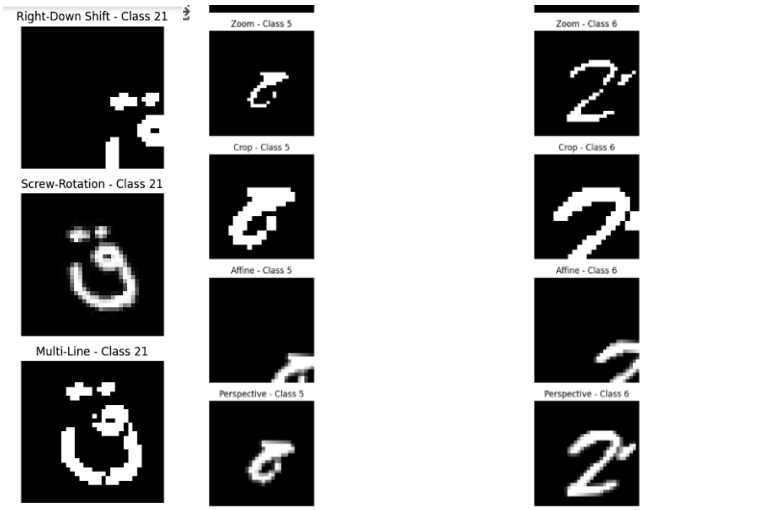
In this section, we systematically explore and visualize various data augmentation methods to inform our decision-making process for selecting the most suitable techniques for our specific data and task. The visualization of these augmentation techniques serves as a crucial step in understanding their impact on the dataset and helps us determine the most effective methods to enhance the robustness and generalization of our model. The following data augmentation techniques have been visualized, each accompanied by images illustrating the transformations applied:

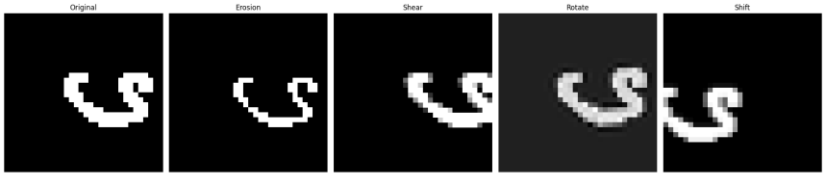
- Erosion: Applying erosion using OpenCV's cv2.erode function with a specified kernel.
- Shear: Utilizing perspective warping with cv2.warpPerspective to induce shear transformations.
- Rotate: Random rotation of images with a variable angle within the range of -5 to 5 degrees.
- Shift: Introducing random pixel shifts using the random_shift function.
- Zoom: Randomly zooming into images with a maximum zoom factor of 0.1 using random_zoom.
- Crop: Applying random cropping to images with a crop size of 0.7 using the random_crop function.
- Affine: Implementing affine transformations with custom scaling, rotation, shear, and translation parameters using affine_transform.
- Perspective: Introducing perspective transformations with an intensity of 0.05 using perspective_transform.
- Blur: Applying Gaussian blur with a sigma value of 2 using gaussian_blur.
- Horizontal Flip: Flipping images horizontally using horizontal_flip.
- Vertical Flip: Flipping images vertically using vertical_flip.
- Right-Down Shift: Shifting images towards the bottom-right corner with shift_right_down.
- Screw-Rotation: Applying screw rotation to images using the respective augmentation method.
- Multi-Line: Implementing multi-line transformations on images for increased variability.

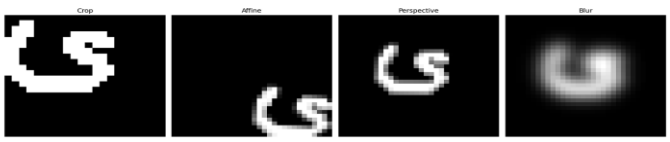
By visually inspecting the effects of these augmentation techniques, we aim to make informed decisions on the selection of methods that best suit the characteristics of our dataset and contribute to the improvement of our model's performance.




Erosion - Class 8
Erosion - Class 5
Erosion - Class 6
Shear - Class 8
Shear - Class 5
Shear - Class 6
Rotate - Class 8
Rotate - Class 5
Rotate - Class 6
Shift - Class 8
Shift - Class 5
Shift - Class 6


Right-Down Shift - Class 21
Zoom - Class 5
Zoom - Class 6
Crop - Class 5
Crop - Class 6
Screw-Rotation - Class 21
Affine - Class 5
Affine - Class 6
Multi-Line - Class 21
Perspective - Class 5
Perspective - Class 6


Original
Erosion
Shear
Rotate
Shift


Crop
Affine
Perspective
Blur

From our comprehensive visualization of various data augmentation methods, we have identified four successful techniques to augment our dataset effectively. We have opted to include the following methods in our augmentation pipeline: a rotation range for diverse orientations, horizontal flipping to create mirror images, a zoom range for varying object scales, and the use of 'nearest' fill mode to maintain pixel integrity. These selected techniques align with the characteristics of our data and are expected to enhance the model's robustness and generalization capabilities. Moving forward, these focused augmentation strategies will play a pivotal role in improving the overall performance of our model.

3. Tasks (Experimental setup and results)

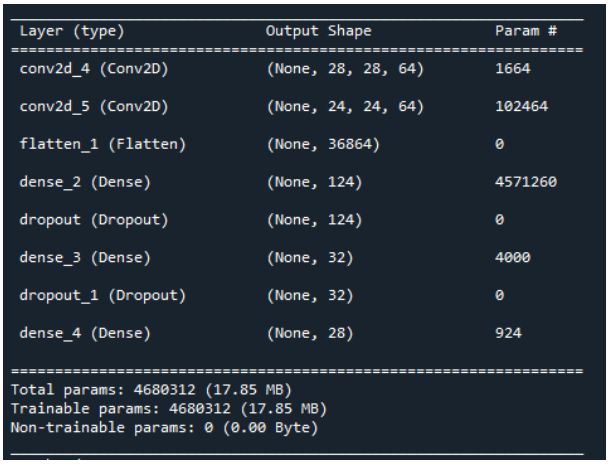
The project comprises several tasks: building and training a custom CNN, employing data augmentation techniques, retraining networks with augmented data, and utilizing pre-trained networks for AHCR.

Task 1&2: Building and Training a Custom CNN for AHCR and doing data augmentation

In pursuit of our task, we have crafted three distinctive Convolutional Neural Network (CNN) models tailored to address the specific requirements of our objective. Each model is designed to capture intricate patterns and features within our dataset. Here is a brief presentation of each custom CNN model:

First model tested

1. **Model summary**



Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 28, 28, 64)	1664
conv2d_5 (Conv2D)	(None, 24, 24, 64)	102464
flatten_1 (Flatten)	(None, 36864)	0
dense_2 (Dense)	(None, 124)	4571260
dropout (Dropout)	(None, 124)	0
dense_3 (Dense)	(None, 32)	4000
dropout_1 (Dropout)	(None, 32)	0
dense_4 (Dense)	(None, 28)	924

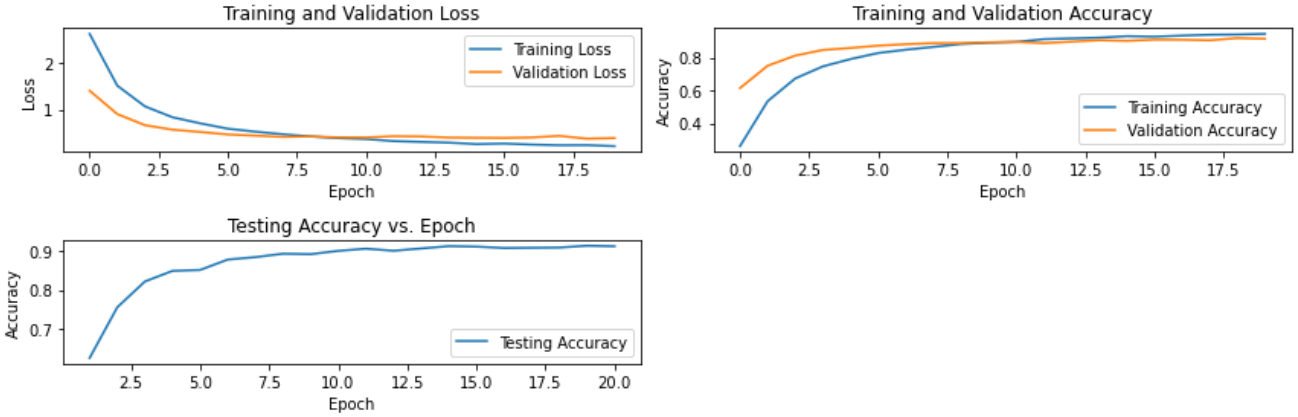
Total params: 4680312 (17.85 MB)
Trainable params: 4680312 (17.85 MB)
Non-trainable params: 0 (0.00 Byte)

Here, we developed a custom (CNN) model using Keras, designed to classify images into 28 unique classes. The model architecture begins with two convolutional layers, each with 64 filters of size 5x5 and ReLU activation, followed by a flattening layer to convert the 2D feature maps into a 1D vector. This is followed by a dense layer with 124 neurons and ReLU activation, incorporating dropout regularization with a rate of 0.3 to prevent overfitting. Another dense layer with 32 neurons follows, using L2 regularization to further combat overfitting, and another

dropout layer with the same rate. The final layer is a dense layer with a softmax activation function to output the probabilities for the number of unique classes.

The model uses the Adam optimizer and is compiled with categorical crossentropy as the loss function and accuracy as the metric for performance evaluation. An early stopping callback is defined to monitor the validation loss and halt training if there isn't an improvement after three epochs, restoring the best weights observed during training to prevent overfitting on the training data.

For training, we split your dataset into a training set and a validation set, with 20% of the data reserved for validation to evaluate the model's performance on unseen data. The model is then trained on the training set with these settings, including our custom callback for additional monitoring or actions during the training process, aiming to optimize performance on the validation set while avoiding overfitting.



1 Required plots

```
Epoch 1/20
335/336 [============================>.] - ETA: 0s - loss: 2.6145 - accuracy: 0.2597
Testing loss: 1.3825, test accuracy: 0.6229
336/336 [==============================] - 19s 54ms/step - loss: 2.6120 - accuracy: 0.2601 - val_loss: 1.3993 - val_accuracy:
0.6138
Epoch 2/20
336/336 [==============================] - ETA: 0s - loss: 1.5076 - accuracy: 0.5339
Testing loss: 0.8986, test accuracy: 0.7548
336/336 [==============================] - 18s 54ms/step - loss: 1.5076 - accuracy: 0.5339 - val_loss: 0.9014 - val_accuracy:
0.7500
Epoch 3/20
336/336 [==============================] - ETA: 0s - loss: 1.0657 - accuracy: 0.6728
Testing loss: 0.6520, test accuracy: 0.8220
336/336 [==============================] - 19s 55ms/step - loss: 1.0657 - accuracy: 0.6728 - val_loss: 0.6637 - val_accuracy:
0.8121
Epoch 4/20
336/336 [==============================] - ETA: 0s - loss: 0.8325 - accuracy: 0.7461
Testing loss: 0.5550, test accuracy: 0.8497
336/336 [==============================] - 18s 53ms/step - loss: 0.8325 - accuracy: 0.7461 - val_loss: 0.5681 - val_accuracy:
0.8464
Epoch 5/20
335/336 [============================>.] - ETA: 0s - loss: 0.7040 - accuracy: 0.7904
Testing loss: 0.5232, test accuracy: 0.8521
336/336 [==============================] - 18s 53ms/step - loss: 0.7031 - accuracy: 0.7908 - val_loss: 0.5193 - val_accuracy:
0.8586
Epoch 6/20
336/336 [==============================] - ETA: 0s - loss: 0.5889 - accuracy: 0.8277
Testing loss: 0.4544, test accuracy: 0.8792
336/336 [==============================] - 18s 53ms/step - loss: 0.5889 - accuracy: 0.8277 - val_loss: 0.4658 - val_accuracy:
0.8728
Epoch 7/20
336/336 [==============================] - ETA: 0s - loss: 0.5250 - accuracy: 0.8479
Testing loss: 0.4392, test accuracy: 0.8854
336/336 [==============================] - 20s 59ms/step - loss: 0.5250 - accuracy: 0.8479 - val_loss: 0.4407 - val_accuracy:
0.8813
Epoch 8/20
```

2 Results_model1

```
Epoch 8/20
336/336 [==============================] - ETA: 0s - loss: 0.4697 - accuracy: 0.8650
Testing loss: 0.4063, test accuracy: 0.8943
336/336 [==============================] - 21s 62ms/step - loss: 0.4697 - accuracy: 0.8650 - val_loss: 0.4133 - val_accuracy:
0.8880
Epoch 9/20
335/336 [============================>.] - ETA: 0s - loss: 0.4229 - accuracy: 0.8826
Testing loss: 0.4299, test accuracy: 0.8932
336/336 [==============================] - 18s 53ms/step - loss: 0.4232 - accuracy: 0.8827 - val_loss: 0.4243 - val_accuracy:
0.8884
Epoch 10/20
335/336 [============================>.] - ETA: 0s - loss: 0.3879 - accuracy: 0.8910
Testing loss: 0.3948, test accuracy: 0.9018
336/336 [==============================] - 18s 54ms/step - loss: 0.3885 - accuracy: 0.8908 - val_loss: 0.4029 - val_accuracy:
0.8929
Epoch 11/20
335/336 [============================>.] - ETA: 0s - loss: 0.3669 - accuracy: 0.8956
Testing loss: 0.3796, test accuracy: 0.9074
336/336 [==============================] - 18s 53ms/step - loss: 0.3670 - accuracy: 0.8955 - val_loss: 0.4018 - val_accuracy:
0.8951
Epoch 12/20
336/336 [==============================] - ETA: 0s - loss: 0.3245 - accuracy: 0.9124
Testing loss: 0.4120, test accuracy: 0.9021
336/336 [==============================] - 18s 55ms/step - loss: 0.3245 - accuracy: 0.9124 - val_loss: 0.4285 - val_accuracy:
0.8880
Epoch 13/20
335/336 [============================>.] - ETA: 0s - loss: 0.3068 - accuracy: 0.9168
Testing loss: 0.4093, test accuracy: 0.9080
336/336 [==============================] - 18s 55ms/step - loss: 0.3070 - accuracy: 0.9169 - val_loss: 0.4224 - val_accuracy:
0.8969
Epoch 14/20
335/336 [============================>.] - ETA: 0s - loss: 0.2913 - accuracy: 0.9211
Testing loss: 0.3845, test accuracy: 0.9140
336/336 [==============================] - 19s 56ms/step - loss: 0.2910 - accuracy: 0.9212 - val_loss: 0.3999 - val_accuracy:
0.9059
Epoch 15/20
335/336 [============================>.] - ETA: 0s - loss: 0.2607 - accuracy: 0.9307
Testing loss: 0.3908, test accuracy: 0.9128
```

3 Results_model1-a-

```
Epoch 15/20
335/336 [============================>.] - ETA: 0s - loss: 0.2607 - accuracy: 0.9307
Testing loss: 0.3908, test accuracy: 0.9128
336/336 [==============================] - 18s 52ms/step - loss: 0.2610 - accuracy: 0.9306 - val_loss: 0.3956 - val_accuracy:
0.9018
Epoch 16/20
335/336 [============================>.] - ETA: 0s - loss: 0.2719 - accuracy: 0.9273
Testing loss: 0.3796, test accuracy: 0.9092
336/336 [==============================] - 18s 53ms/step - loss: 0.2717 - accuracy: 0.9275 - val_loss: 0.3923 - val_accuracy:
0.9092
Epoch 17/20
336/336 [==============================] - ETA: 0s - loss: 0.2500 - accuracy: 0.9341
Testing loss: 0.3995, test accuracy: 0.9098
336/336 [==============================] - 18s 53ms/step - loss: 0.2500 - accuracy: 0.9341 - val_loss: 0.4026 - val_accuracy:
0.9089
Epoch 18/20
336/336 [==============================] - ETA: 0s - loss: 0.2357 - accuracy: 0.9392
Testing loss: 0.4222, test accuracy: 0.9104
336/336 [==============================] - 18s 52ms/step - loss: 0.2357 - accuracy: 0.9392 - val_loss: 0.4346 - val_accuracy:
0.9055
Epoch 19/20
336/336 [==============================] - ETA: 0s - loss: 0.2375 - accuracy: 0.9408
Testing loss: 0.3831, test accuracy: 0.9152
336/336 [==============================] - 18s 54ms/step - loss: 0.2375 - accuracy: 0.9408 - val_loss: 0.3763 - val_accuracy:
0.9185
Epoch 20/20
335/336 [============================>.] - ETA: 0s - loss: 0.2170 - accuracy: 0.9447
Testing loss: 0.3754, test accuracy: 0.9137
336/336 [==============================] - 20s 58ms/step - loss: 0.2169 - accuracy: 0.9447 - val_loss: 0.3895 - val_accuracy:
0.9148
105/105 [==============================] - 1s 10ms/step - loss: 0.3754 - accuracy: 0.9137
Test accuracy: 91.37%
```

4 Results_model1-b-

Based on the provided plots and training logs, here's a detailed explanation of this CNN model's results:

The training loss decreases steadily as the epochs increase, which is a good indication that the model is learning and improving from the training data. The validation loss also decreases and follows the training loss closely but with a slight gap, which suggests that the model is generalizing well and not overfitting significantly.

Both the training and validation accuracy increase over time. The training accuracy is consistently higher than the validation accuracy, which is expected because the model is directly learning from the training data. The validation accuracy is relatively high as well, which indicates that the model's improvements are translating well to unseen data.

The testing accuracy plot shows a steady increase in accuracy over epochs. This demonstrates that the model's ability to generalize to new, unseen data is improving as it continues to learn.

The logs provide detailed information for each epoch, including training and testing loss and accuracy. The testing accuracy increases with each epoch, reaching a high of around 91.4% by the final epoch. This high testing accuracy is a strong indicator that the model has learned to classify the images with high reliability.

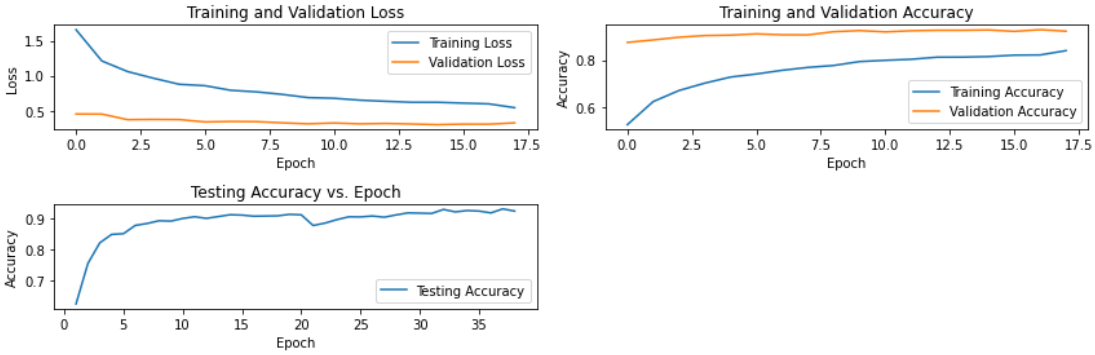
The model does not show signs of overfitting, which is evident from the close following of the training and validation loss/accuracy curves. Early stopping did not trigger, as the validation loss continued to improve or remain stable, indicating that the chosen patience level was appropriate. The final testing accuracy of approximately 91.4% suggests that the model is performing well on the test set, which is a good indicator of its ability to generalize. The model's performance could potentially be improved with further hyperparameter tuning, data augmentation, or by adding more complexity to the model.

Results after Data Augmentation:

As mentioned before, the following methods in our augmentation pipeline were used: a rotation range for diverse orientations, horizontal flipping to create mirror images, a zoom range for varying object scales, and the use of 'nearest' fill mode to maintain pixel integrity.

```
# Create an ImageDataGenerator
datagen = ImageDataGenerator(
    rotation_range=10,
    horizontal_flip=True,
    zoom_range=0.2,
    fill_mode='nearest'
)
```

5 Methods in our augmentation pipeline



6 Required plots

```
Epoch 1/20
335/336 [============================>.] - ETA: 0s - loss: 1.6566 - accuracy: 0.5259
Testing loss: 0.4669, test accuracy: 0.8789
336/336 [==============================] - 19s 54ms/step - loss: 1.6560 - accuracy: 0.5259 - val_loss: 0.4529 - val_accuracy:
0.8757
Epoch 2/20
335/336 [============================>.] - ETA: 0s - loss: 1.2108 - accuracy: 0.6238
Testing loss: 0.4400, test accuracy: 0.8866
336/336 [==============================] - 18s 53ms/step - loss: 1.2102 - accuracy: 0.6241 - val_loss: 0.4518 - val_accuracy:
0.8862
Epoch 3/20
335/336 [============================>.] - ETA: 0s - loss: 1.0583 - accuracy: 0.6712
Testing loss: 0.3746, test accuracy: 0.8979
336/336 [==============================] - 18s 54ms/step - loss: 1.0581 - accuracy: 0.6711 - val_loss: 0.3719 - val_accuracy:
0.8977
Epoch 4/20
336/336 [==============================] - ETA: 0s - loss: 0.9636 - accuracy: 0.7025
Testing loss: 0.3743, test accuracy: 0.9071
336/336 [==============================] - 18s 54ms/step - loss: 0.9636 - accuracy: 0.7025 - val_loss: 0.3754 - val_accuracy:
0.9048
Epoch 5/20
335/336 [============================>.] - ETA: 0s - loss: 0.8763 - accuracy: 0.7285
Testing loss: 0.3668, test accuracy: 0.9068
336/336 [==============================] - 19s 56ms/step - loss: 0.8769 - accuracy: 0.7284 - val_loss: 0.3730 - val_accuracy:
0.9066
Epoch 6/20
336/336 [==============================] - ETA: 0s - loss: 0.8569 - accuracy: 0.7414
Testing loss: 0.3475, test accuracy: 0.9098
336/336 [==============================] - 18s 55ms/step - loss: 0.8569 - accuracy: 0.7414 - val_loss: 0.3395 - val_accuracy:
0.9118
Epoch 7/20
335/336 [============================>.] - ETA: 0s - loss: 0.7912 - accuracy: 0.7572
Testing loss: 0.3324, test accuracy: 0.9060
336/336 [==============================] - 18s 55ms/step - loss: 0.7911 - accuracy: 0.7572 - val_loss: 0.3461 - val_accuracy:
0.9081
Epoch 8/20
336/336 [==============================] - ETA: 0s - loss: 0.7687 - accuracy: 0.7693
Testing loss: 0.3343, test accuracy: 0.9134
336/336 [==============================] - 18s 55ms/step - loss: 0.7687 - accuracy: 0.7693 - val_loss: 0.3434 - val_accuracy:
0.9077
Epoch 9/20
336/336 [==============================] - ETA: 0s - loss: 0.7324 - accuracy: 0.7776
Testing loss: 0.3130, test accuracy: 0.9199
336/336 [==============================] - 18s 55ms/step - loss: 0.7324 - accuracy: 0.7776 - val_loss: 0.3266 - val_accuracy:
0.9211
Epoch 10/20
```

7 Results_model1_Aug

```
Epoch 9/20
336/336 [==============================] - ETA: 0s - loss: 0.7324 - accuracy: 0.7776
Testing loss: 0.3130, test accuracy: 0.9199
336/336 [==============================] - 18s 55ms/step - loss: 0.7324 - accuracy: 0.7776 - val_loss: 0.3266 - val_accuracy: 0.9211
Epoch 10/20
336/336 [==============================] - ETA: 0s - loss: 0.6868 - accuracy: 0.7942
Testing loss: 0.3109, test accuracy: 0.9190
336/336 [==============================] - 19s 56ms/step - loss: 0.6868 - accuracy: 0.7942 - val_loss: 0.3126 - val_accuracy: 0.9260
Epoch 11/20
335/336 [============================>.] - ETA: 0s - loss: 0.6773 - accuracy: 0.7995
Testing loss: 0.3094, test accuracy: 0.9182
336/336 [==============================] - 19s 55ms/step - loss: 0.6773 - accuracy: 0.7996 - val_loss: 0.3246 - val_accuracy: 0.9204
Epoch 12/20
336/336 [==============================] - ETA: 0s - loss: 0.6505 - accuracy: 0.8039
Testing loss: 0.2909, test accuracy: 0.9312
336/336 [==============================] - 18s 55ms/step - loss: 0.6505 - accuracy: 0.8039 - val_loss: 0.3122 - val_accuracy: 0.9252
Epoch 13/20
335/336 [============================>.] - ETA: 0s - loss: 0.6341 - accuracy: 0.8131
Testing loss: 0.3160, test accuracy: 0.9232
336/336 [==============================] - 18s 54ms/step - loss: 0.6343 - accuracy: 0.8129 - val_loss: 0.3177 - val_accuracy: 0.9271
Epoch 14/20
335/336 [============================>.] - ETA: 0s - loss: 0.6199 - accuracy: 0.8137
Testing loss: 0.2972, test accuracy: 0.9277
336/336 [==============================] - 19s 55ms/step - loss: 0.6204 - accuracy: 0.8132 - val_loss: 0.3099 - val_accuracy: 0.9271
Epoch 15/20
335/336 [============================>.] - ETA: 0s - loss: 0.6197 - accuracy: 0.8157
Testing loss: 0.2976, test accuracy: 0.9259
336/336 [==============================] - 19s 55ms/step - loss: 0.6199 - accuracy: 0.8155 - val_loss: 0.3006 - val_accuracy: 0.9286
Epoch 16/20
335/336 [============================>.] - ETA: 0s - loss: 0.6086 - accuracy: 0.8215
Testing loss: 0.3178, test accuracy: 0.9199
336/336 [==============================] - 19s 57ms/step - loss: 0.6081 - accuracy: 0.8215 - val_loss: 0.3084 - val_accuracy: 0.9226
Epoch 17/20
335/336 [============================>.] - ETA: 0s - loss: 0.5969 - accuracy: 0.8223
Testing loss: 0.3048, test accuracy: 0.9330
336/336 [==============================] - 18s 55ms/step - loss: 0.5974 - accuracy: 0.8224 - val_loss: 0.3076 - val_accuracy: 0.9297
Epoch 18/20
336/336 [==============================] - ETA: 0s - loss: 0.5429 - accuracy: 0.8407 Restoring model weights from the end of the best epoch: 15.

Testing loss: 0.2976, test accuracy: 0.9259
336/336 [==============================] - 18s 54ms/step - loss: 0.5429 - accuracy: 0.8407 - val_loss: 0.3263 - val_accuracy: 0.9234
Epoch 18: early stopping
105/105 [==============================] - 1s 10ms/step - loss: 0.2976 - accuracy: 0.9259
Test accuracy: 92.59%
```

8 Results_model1_Aug -b-

After implementing data augmentation, the curves for training and validation loss are close, with the validation loss slightly higher than the training loss but following the same downward trend. This is a positive sign, as it suggests that the model is learning effectively and the added variation from augmentation is helping to prevent overfitting.The accuracy curves for both training and validation are quite close and demonstrate a steady increase. This indicates that the model's ability to generalize is improving alongside learning from the augmented data. The testing accuracy continues to increase over the epochs, showing that the model is consistently improving its performance on unseen data. The final testing accuracy is high, suggesting that data augmentation has had a beneficial effect. The logs confirm that the model's performance on both the training and testing sets is improving. Early stopping is triggered at epoch 20, and the best weights are restored from epoch 15, indicating that the model's performance on the validation set started to plateau, which is a built-in mechanism to prevent overfitting.

From these results, we can conclude the following:

- Data augmentation has likely contributed to the model's improved generalization, as indicated by the stable and close loss and accuracy curves between the training and validation sets.

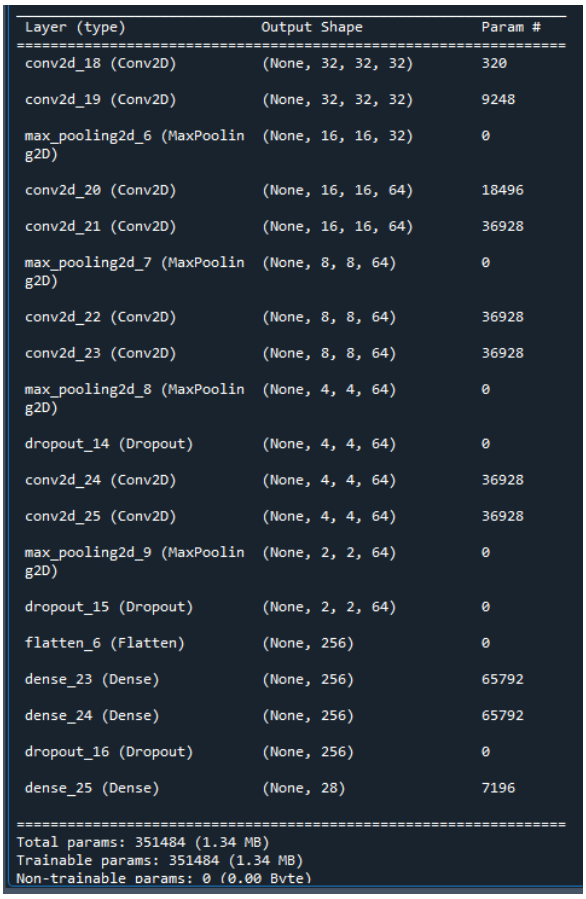
- The early stopping callback has functioned correctly to prevent overfitting by restoring the weights from the best-performing epoch on the validation data.

- A high testing accuracy post-augmentation indicates that the model is robust and performs well on new, unseen data, which is the ultimate goal of a generalizable machine learning model.

Overall, the results post-augmentation are positive, and there are no significant signs of overfitting. The model seems to have benefited from the augmented data, showing improved performance and robustness.

Second model tested

1. Model summary



Layer (type)	Output Shape	Param #
conv2d_18 (Conv2D)	(None, 32, 32, 32)	320
conv2d_19 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_6 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_20 (Conv2D)	(None, 16, 16, 64)	18496
conv2d_21 (Conv2D)	(None, 16, 16, 64)	36928
max_pooling2d_7 (MaxPooling2D)	(None, 8, 8, 64)	0
conv2d_22 (Conv2D)	(None, 8, 8, 64)	36928
conv2d_23 (Conv2D)	(None, 8, 8, 64)	36928
max_pooling2d_8 (MaxPooling2D)	(None, 4, 4, 64)	0
dropout_14 (Dropout)	(None, 4, 4, 64)	0
conv2d_24 (Conv2D)	(None, 4, 4, 64)	36928
conv2d_25 (Conv2D)	(None, 4, 4, 64)	36928
max_pooling2d_9 (MaxPooling2D)	(None, 2, 2, 64)	0
dropout_15 (Dropout)	(None, 2, 2, 64)	0
flatten_6 (Flatten)	(None, 256)	0
dense_23 (Dense)	(None, 256)	65792
dense_24 (Dense)	(None, 256)	65792
dropout_16 (Dropout)	(None, 256)	0
dense_25 (Dense)	(None, 28)	7196

Total params: 351484 (1.34 MB)
Trainable params: 351484 (1.34 MB)
Non-trainable params: 0 (0.00 Byte)

9 **Model2_summary**

About this architecture:

Conv2D Layers:

These layers apply a number of filters to the input image to create feature maps. The first Conv2D layer has 32 filters (hence the 32 in the output shape), and since the input is likely an image, each filter has a small receptive field (commonly 3x3 or 5x5 pixels). The number of parameters in a Conv2D layer is calculated as (filter_height * filter_width * input_channels + 1) * number_of_filters. The "+1" accounts for the bias term for each filter. The first Conv2D layer typically has fewer parameters because it's dealing with raw image data, which has a lower dimensionality.

MaxPooling2D Layers:

These perform down-sampling by dividing the input into rectangular pooling regions and outputting the maximum value of each region. This reduces the spatial resolution of the feature map, reducing the number of parameters and computation in the network, and also helps to make the detection of features invariant to scale and orientation changes. MaxPooling layers have no learnable parameters.

Additional Conv2D Layers:

As we go deeper into the network, we typically increase the number of filters because the network starts to combine the simple features from the initial layers into more complex features. In this network, after each pooling layer, the number of filters doubles. This is a common practice as the spatial resolution decreases, allowing the network to represent more complex features.

Dropout Layers:

These are a form of regularization. By randomly setting a fraction of input units to 0 at each update during training, they prevent overfitting. Dropout layers also have no learnable parameters.

Flatten Layer:

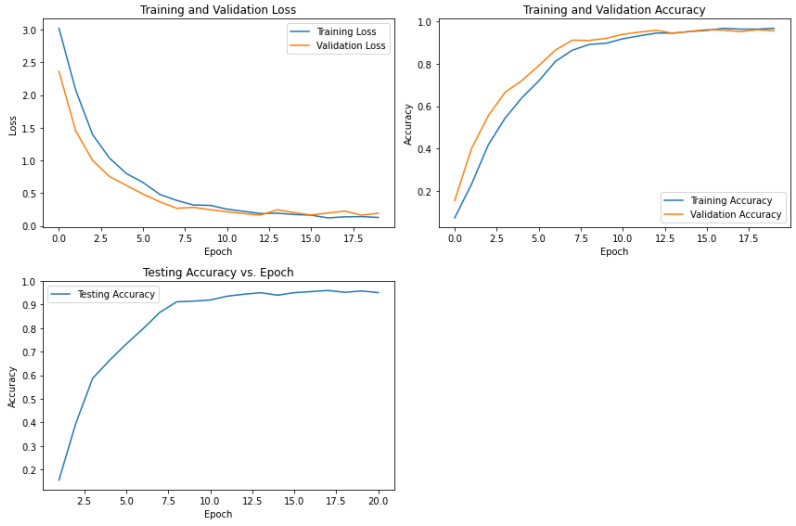
The Flatten layer converts the two-dimensional feature maps into a one-dimensional vector. This is necessary because fully connected layers (Dense layers) expect 1D input. There are no parameters to learn in this layer; it just reformats the data.

Dense Layers:

These layers are fully connected layers that take the flattened feature maps and perform classification.

Final Dense Layer:

The last Dense layer has 28 units, which corresponds to the number of classes the network is trying to predict.



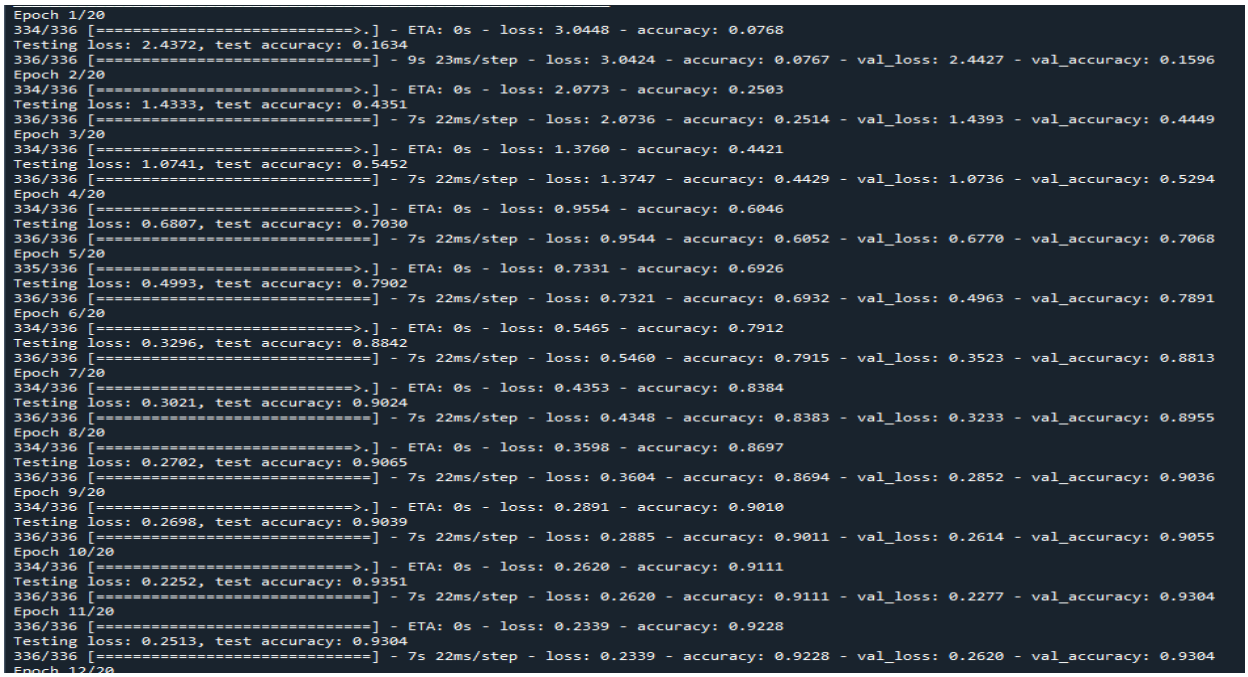
10 **Model2 Plots**

Training and Validation Loss: This graph shows the loss of the model on the training set (blue line) and the validation set (orange line) as a function of the epoch (the number of times the entire training dataset is passed through the model). As the number of epochs increases, both training and validation loss decrease, which is indicative of learning. However, the training loss decreases

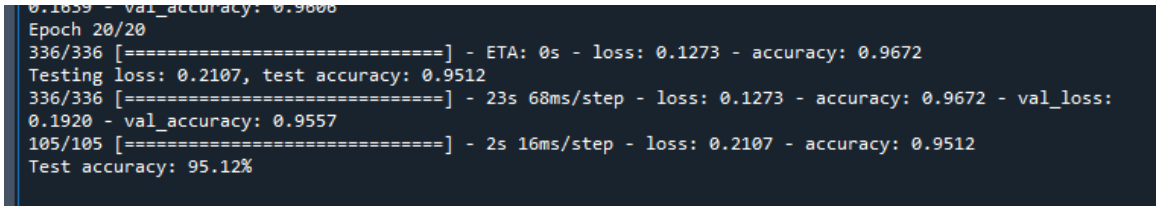
faster and is lower than the validation loss, which can be a sign of the model starting to overfit the training data.

Training and Validation Accuracy : This graph shows the accuracy of the model on the training set (blue line) and the validation set (orange line) over the epochs. Both accuracies improve over time, and they remain close to each other, which is generally a good sign that the model is generalizing well and not overfitting significantly.

Testing Accuracy vs. Epoch: This graph plots the testing accuracy alone over the epochs. It shows a rapid increase in the initial epochs and then plateaus, which is typical as the model begins to converge to its best performance.



11 Model2 Results -a-



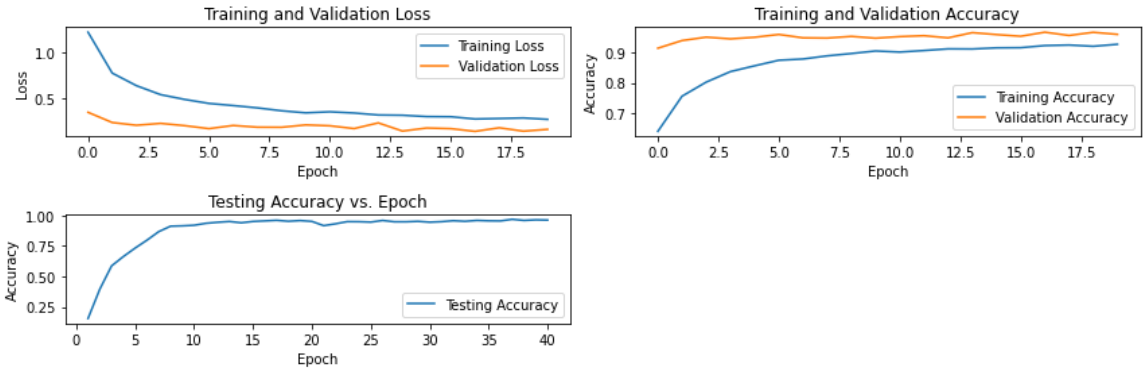
12 Model2 Results -b-

The model is trained for 20 epochs, and we can see a consistent improvement in loss and accuracy over time. Notably, in the final epoch (Epoch 20/20), the testing loss is around 0.1273, and the testing accuracy is 96.72%, which is then followed by a separate final test accuracy calculation of 95.12%. This slight difference could be due to variations in the data subsets used

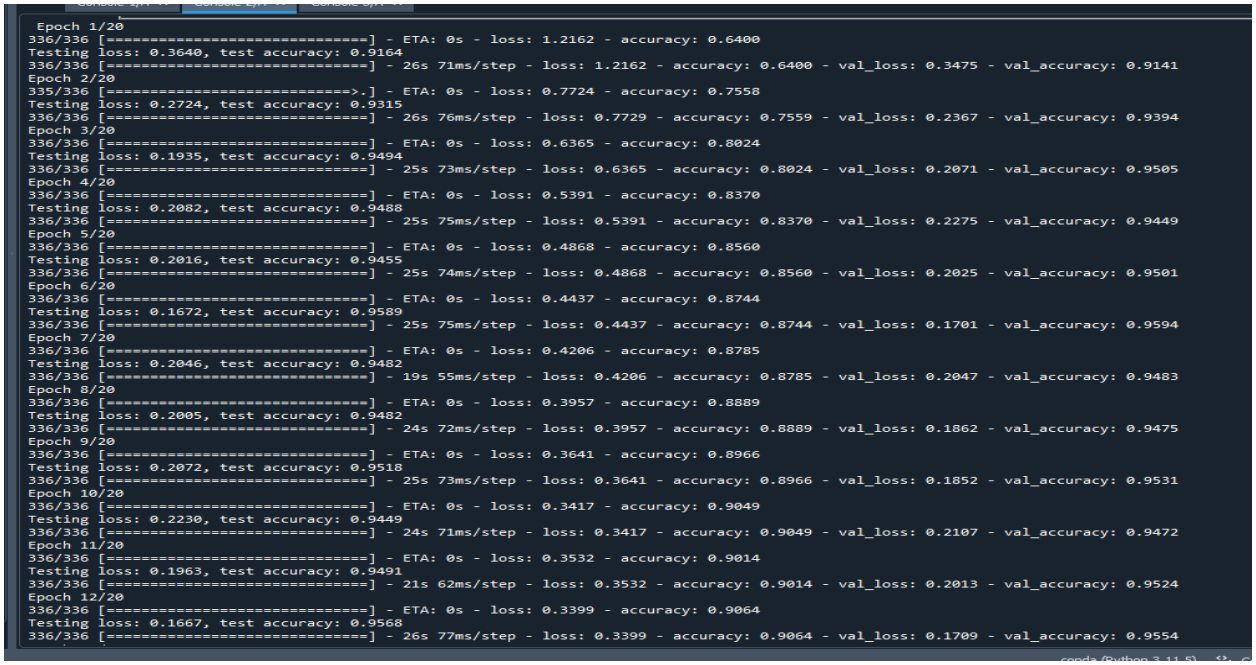
for the final test or a small overfitting that doesn't affect the validation but impacts the test slightly.

Both training and validation loss decrease as the number of epochs increases, which is a good sign that the model is learning and improving. the training and validation accuracy, both increasing over time, which is also positive. testing accuracy improving as the number of epochs increases, suggesting that the model's ability to generalize to new data is improving. The model is performing well, as evidenced by the high accuracy and low loss on both validation and testing datasets. There might be a slight overfitting, as suggested by the consistently lower training loss compared to the validation loss. However, the accuracies are close, suggesting that if overfitting is present, it's not significantly detrimental to the model's performance on unseen data.

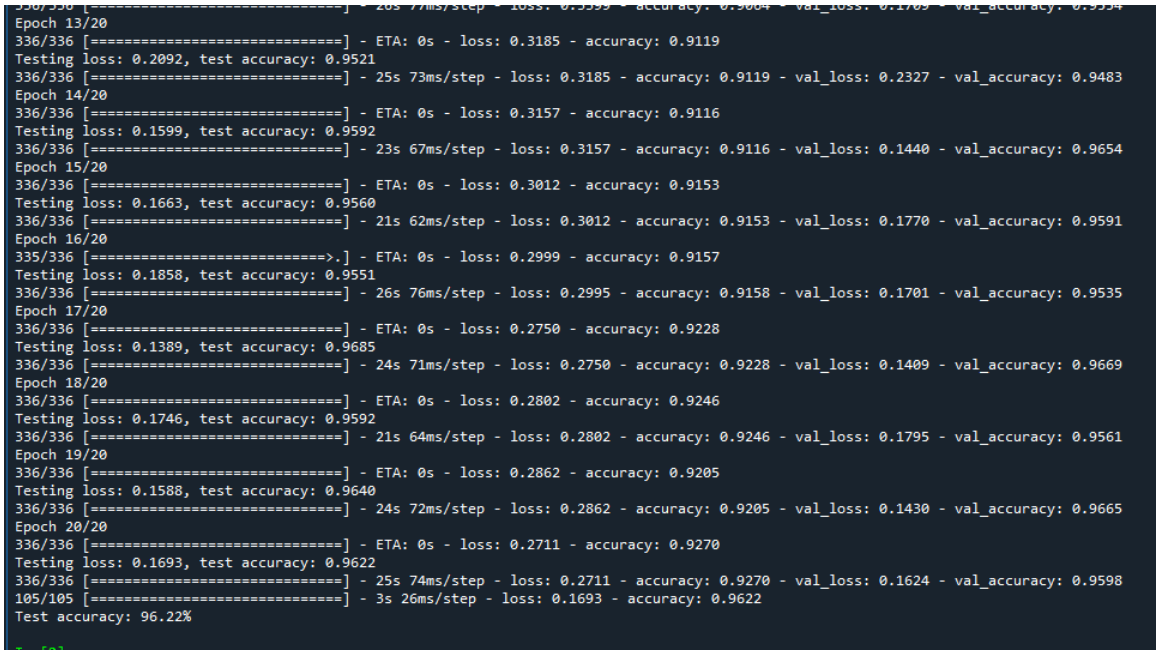
Results after Data Augmentation:



13Model2 Plots after Aug



14 Results After Augmentation -a-



15 Results After Augmenation-b-

We can see that the training loss decreases rapidly and then plateaus, which is typical and desired in a training process. The validation loss decreases alongside the training loss, which suggests that the model is generalizing well and not overfitting.

The training accuracy increases over time, as expected. The validation accuracy also increases and follows the training accuracy closely, which is a good sign that the model is performing consistently on both the training and unseen validation data.

The testing accuracy increases sharply in the first few epochs and then begins to plateau, indicating that most learning occurs early, and additional epochs provide diminishing returns on model performance.

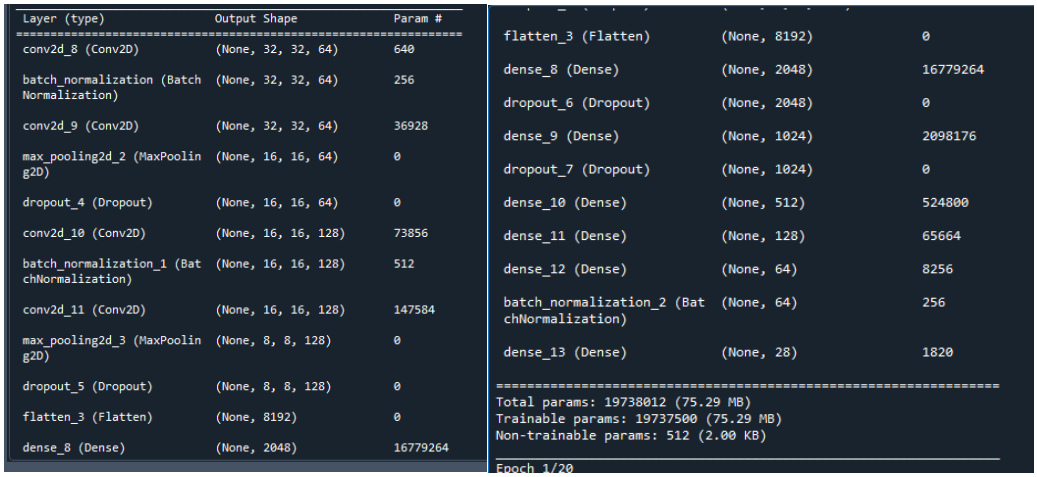
The final test accuracy is reported as 96.22%, which is an improvement over the 95.12% test accuracy reported before augmentation.

The graphs indicate that the model's learning has stabilized after data augmentation, with less volatility in the loss and accuracy curves. This stabilization is a common effect of data augmentation, as it helps the model learn from a more diverse dataset, reducing overfitting and improving generalization.

Overall, the augmentation process seems to have had a positive effect on the model's ability to generalize and perform on unseen data, as evidenced by the reduced loss and increased accuracy on both validation and testing datasets.

Third model tested

1. Model summary



Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 32, 32, 64)	640
batch_normalization (BatchNormalization)	(None, 32, 32, 64)	256
conv2d_9 (Conv2D)	(None, 32, 32, 64)	36928
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 64)	0
dropout_4 (Dropout)	(None, 16, 16, 64)	0
conv2d_10 (Conv2D)	(None, 16, 16, 128)	73856
batch_normalization_1 (BatchNormalization)	(None, 16, 16, 128)	512
conv2d_11 (Conv2D)	(None, 16, 16, 128)	147584
max_pooling2d_3 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_5 (Dropout)	(None, 8, 8, 128)	0
flatten_3 (Flatten)	(None, 8192)	0
dense_8 (Dense)	(None, 2048)	16779264
dropout_6 (Dropout)	(None, 2048)	0
dense_9 (Dense)	(None, 1024)	2098176
dropout_7 (Dropout)	(None, 1024)	0
dense_10 (Dense)	(None, 512)	524800
dense_11 (Dense)	(None, 128)	65664
dense_12 (Dense)	(None, 64)	8256
batch_normalization_2 (BatchNormalization)	(None, 64)	256
dense_13 (Dense)	(None, 28)	1820

Total params: 19738012 (75.29 MB)
Trainable params: 19737500 (75.29 MB)
Non-trainable params: 512 (2.00 KB)

Epoch 1/20

16 Model Summary

Notable components and their parameters:

Convolutional Layers (conv2d_8, conv2d_9, etc.): These are the core building blocks of a ConvNet that perform convolutional operations on the input data. They capture spatial hierarchies and features from the input images. The parameters in these layers are the weights of the filters that are learned during training. For example, conv2d_9 has 36,928 parameters, indicating that there are numerous filters each learning different features.

Batch Normalization (batch_normalization, batch_normalization_1, etc.): These layers normalize the inputs to each layer, which helps in speeding up the training and provides some regularization effect. They have a relatively small number of parameters, as they are only scaling and shifting the data.

Max Pooling Layers (max_pooling2d_2, max_pooling2d_3, etc.): These layers reduce the spatial dimensions of the input, which decreases the number of parameters, computation, and helps in making the detection of features invariant to scale and orientation changes.

Dense Layers (dense_8, dense_9, etc.): Also known as fully connected layers, these are where most of the network parameters reside. For instance, dense_8 has a staggering 16,779,264 parameters, indicating it is a major site for learning during the network's training.

Dropout Layers (dropout_4, dropout_5, etc.): These layers randomly set a fraction of input units to 0 at each update during training time, which helps prevent overfitting. They do not add to the parameter count since they do not have learnable parameters.

Flatten Layer (flatten_3): This layer does not have parameters as it only reshapes the data, converting the 2D feature maps into a 1D feature vector to be fed into the dense layers.

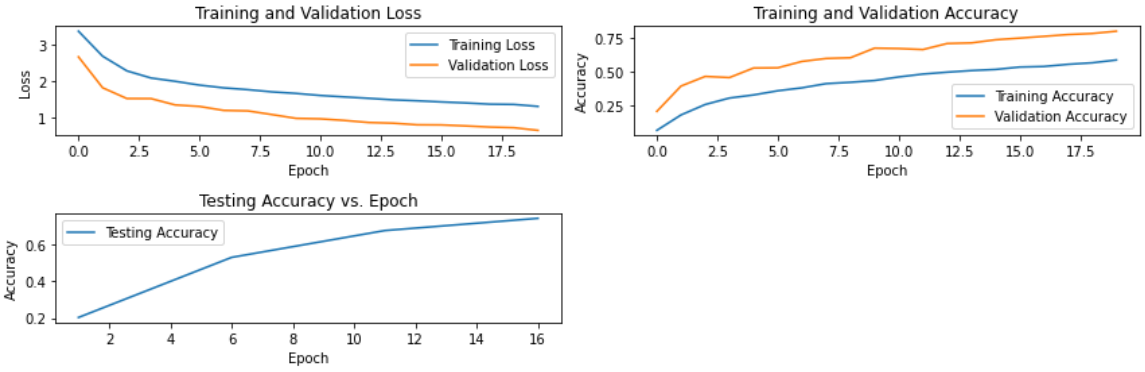
Final Dense Layer (dense_13): The last dense layer in the network typically represents the output layer, which in this case has 28 units. It may correspond to a classification task with 28 classes.

17 Required Plots for Model3

```
0.9386
Epoch 13: early stopping
105/105 [==============================] - 2s 22ms/step - loss: 0.8560 - accuracy: 0.9354
Test accuracy: 93.54%
```

18 Results for Model3

Results after Augmentation:



19 Required Plots for Model3 after augmentation

```
Epoch 13: early stopping
105/105 [==============================] - 3s 28ms/step - loss: 1.0176 - accuracy: 0.9452
Test accuracy: 94.52%
```

20 Results for Model3 after augmentation

The training loss decreases sharply at the beginning, indicating rapid learning, and then starts to plateau, showing that the model is starting to converge. The validation loss decreases alongside the training loss but starts to plateau and diverges slightly upwards towards the end of the epochs. This could be a sign of the model beginning to overfit.

Both training and validation accuracy increase rapidly and then level off, with training accuracy being slightly higher than validation accuracy throughout. This is expected behavior as the model will typically perform better on data it has seen (training data) compared to new data (validation data).

The testing accuracy increases significantly in the first few epochs, which indicates that the model is learning generalizable features from the training data. After the initial jump, the testing accuracy continues to increase more gradually.

The training process has been stopped early at epoch 13, as indicated by the "early stopping" note. The reported test accuracy is 93.54%.

The results from this architecture show that the model is capable of learning and generalizing from the data well, achieving a high test accuracy. The use of early stopping suggests an effort to prevent overfitting, which seems to be successful given the high test accuracy. However, there is a slight indication of overfitting as seen by the divergence of training and validation loss, which could be further investigated and mitigated with techniques such as further data augmentation, regularization, or tuning the dropout rate.

Comparison between our three custom models

Model 1:

A custom CNN model with a simpler architecture. Demonstrated a good balance between training and validation performance, indicating effective learning without significant overfitting. Achieved a test accuracy of approximately 91.4% before data augmentation. After data augmentation, the model showed improved generalization with slightly higher test accuracy, and early stopping was appropriately used to prevent overfitting.

Model 2:

This model had a more complex architecture with a higher number of parameters, particularly in the dense layers. It also showed learning without major signs of overfitting, with a close match between training and validation loss and accuracy. The test accuracy before augmentation was 95.12%, which increased to 96.22% after augmentation, indicating a beneficial effect of the augmentation process on the model's ability to generalize.

Model 3:

The architecture of this model was notable for its deep sequence of layers, including multiple dense layers with a very high number of parameters. Early stopping was used to prevent overfitting, with the model achieving a test accuracy of 93.54%. Similar to the other models, there was a slight divergence between training and validation loss, potentially indicating the beginning of overfitting, which early stopping helped mitigate.

Task 3: Utilizing Published CNN Architectures

Model chosen Alexnet

AlexNet is a pioneering (CNN) architecture that played a pivotal role in the advancement of deep learning for computer vision tasks. Introduced by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton in 2012, AlexNet gained widespread recognition for winning the ImageNet Large Scale

Visual Recognition Challenge (ILSVRC) with a significant margin. Its architecture comprises five convolutional layers, followed by max-pooling layers and three fully connected layers. The use of (ReLU) as activation functions and dropout layers for regularization contributed to its success. The decision to choose AlexNet is rooted in its ability to capture complex hierarchical features in images efficiently, making it well-suited for image classification tasks.

Layer (type)	Output Shape	Param #
resizing_8 (Resizing)	(None, 224, 224, 1)	0
conv2d_56 (Conv2D)	(None, 56, 56, 96)	11712
lambda_16 (Lambda)	(None, 56, 56, 96)	0
activation_40 (Activation)	(None, 56, 56, 96)	0
max_pooling2d_24 (MaxPooling2D)	(None, 27, 27, 96)	0
conv2d_57 (Conv2D)	(None, 7, 7, 256)	614656
lambda_17 (Lambda)	(None, 7, 7, 256)	0
activation_41 (Activation)	(None, 7, 7, 256)	0
max_pooling2d_25 (MaxPooling2D)	(None, 3, 3, 256)	0
conv2d_58 (Conv2D)	(None, 1, 1, 384)	885120
activation_42 (Activation)	(None, 1, 1, 384)	0
conv2d_59 (Conv2D)	(None, 1, 1, 384)	1327488
activation_43 (Activation)	(None, 1, 1, 384)	0
conv2d_60 (Conv2D)	(None, 1, 1, 256)	884992
activation_44 (Activation)	(None, 1, 1, 256)	0
flatten_11 (Flatten)	(None, 256)	0
dense_32 (Dense)	(None, 4096)	1052672
dropout_23 (Dropout)	(None, 4096)	0
dense_33 (Dense)	(None, 4096)	16781312
dense_33 (Dense)	(None, 4096)	16781312
dropout_24 (Dropout)	(None, 4096)	0
dense_34 (Dense)	(None, 28)	114716

Total params: 21672668 (82.67 MB)
Trainable params: 21672668 (82.67 MB)
Non-trainable params: 0 (0.00 Byte)

21 Task3 Model summary

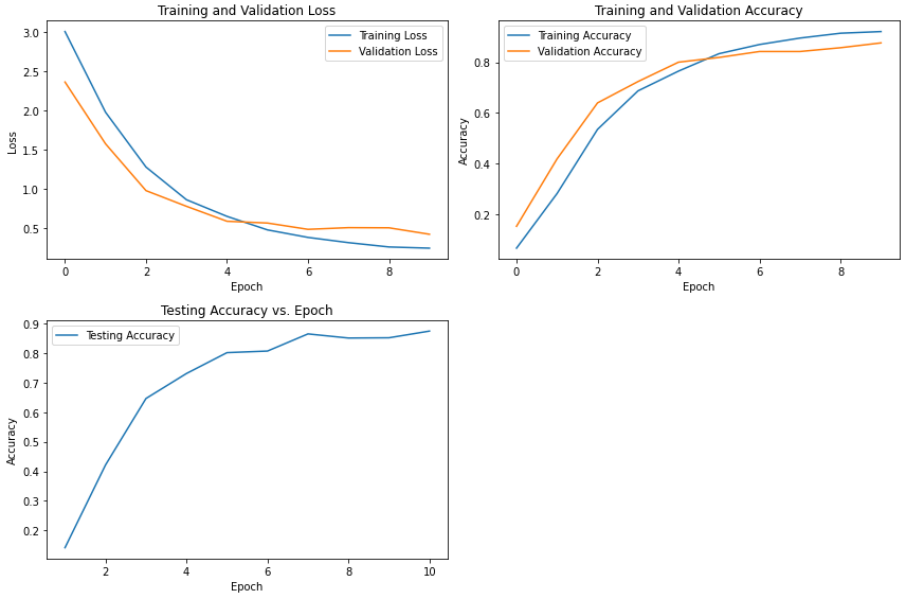
The fine-tuning adjustments in our network architecture, compared to the original AlexNet, encompass various key elements. Beginning with the Input Layer, while AlexNet typically processes images with dimensions of 227x227x3, our network adopts an input layer with an image size of 224x224x1. Moving to the Convolutional Layers, alterations are observed in both the number and size of filters. Unlike AlexNet's use of 11x11, 5x5, and 3x3 filters with varying quantities (96, 256, 384, etc.), our network may employ different configurations. In the realm of Fully Connected Layers (Dense), AlexNet features three layers with 4096, 4096, and 1000 units, respectively. Conversely, our network's final dense layer is comprised of 28 units, reflecting fine-tuning on a dataset housing 28 classes. Additionally, we integrate Dropout Layers to mitigate overfitting by randomly deactivating units and their connections during training. These nuanced adjustments collectively contribute to the tailored architecture of our fine-tuned network.

Training before Augmentation (on 20 epochs)

22 Task3 Training before Augmentation (on 20 epochs)

```
Epoch 9/20
336/336 [==============================] - 68s 202ms/step - loss: 0.4450 - accuracy: 0.8490 - val_loss: 0.6431 - val_accuracy: 0.7946
Epoch 10/20
336/336 [==============================] - 69s 206ms/step - loss: 0.3619 - accuracy: 0.8786 - val_loss: 0.5996 - val_accuracy: 0.8036
Epoch 11/20
336/336 [==============================] - 72s 214ms/step - loss: 0.3357 - accuracy: 0.8876 - val_loss: 0.5251 - val_accuracy: 0.8367
Epoch 12/20
336/336 [==============================] - 70s 208ms/step - loss: 0.2697 - accuracy: 0.9117 - val_loss: 0.5510 - val_accuracy: 0.8512
Epoch 13/20
336/336 [==============================] - 71s 210ms/step - loss: 0.2374 - accuracy: 0.9250 - val_loss: 0.6036 - val_accuracy: 0.8363
Epoch 14/20
336/336 [==============================] - 69s 205ms/step - loss: 0.2209 - accuracy: 0.9289 - val_loss: 0.5501 - val_accuracy: 0.8523
Epoch 15/20
336/336 [==============================] - 70s 209ms/step - loss: 0.1924 - accuracy: 0.9392 - val_loss: 0.5423 - val_accuracy: 0.8575
Epoch 16/20
336/336 [==============================] - 70s 207ms/step - loss: 0.1852 - accuracy: 0.9426 - val_loss: 0.6252 - val_accuracy: 0.8408
Epoch 17/20
336/336 [==============================] - 71s 213ms/step - loss: 0.1661 - accuracy: 0.9487 - val_loss: 0.6048 - val_accuracy: 0.8571
Epoch 18/20
336/336 [==============================] - 74s 220ms/step - loss: 0.1331 - accuracy: 0.9594 - val_loss: 0.5973 - val_accuracy: 0.8709
Epoch 19/20
336/336 [==============================] - 70s 209ms/step - loss: 0.1649 - accuracy: 0.9534 - val_loss: 0.5816 - val_accuracy: 0.8523
Epoch 20/20
336/336 [==============================] - 72s 214ms/step - loss: 0.1273 - accuracy: 0.9617 - val_loss: 0.5863 - val_accuracy: 0.8798
105/105 [==============================] - 4s 40ms/step - loss: 0.5139 - accuracy: 0.8863
Test accuracy: 88.63%

In [19]:
```

23 Task3- Results of Training before Augmentation (on 20 epochs)



24 Task3 on ten epochs before augmentation

```
336/336 [==============================] - ETA: 0s - loss: 3.0014 - accuracy: 0.0679
Testing loss: 2.3762, test accuracy: 0.1417
336/336 [==============================] - 79s 230ms/step - loss: 3.0014 - accuracy: 0.0679 - val_loss: 2.3594 - val_accuracy: 0.1540
Epoch 2/10
336/336 [==============================] - ETA: 0s - loss: 1.9758 - accuracy: 0.2828
Testing loss: 1.5686, test accuracy: 0.4214
336/336 [==============================] - 78s 231ms/step - loss: 1.9758 - accuracy: 0.2828 - val_loss: 1.5737 - val_accuracy: 0.4193
Epoch 3/10
336/336 [==============================] - ETA: 0s - loss: 1.2775 - accuracy: 0.5356
Testing loss: 0.9688, test accuracy: 0.6464
336/336 [==============================] - 77s 230ms/step - loss: 1.2775 - accuracy: 0.5356 - val_loss: 0.9775 - val_accuracy: 0.6399
Epoch 4/10
336/336 [==============================] - ETA: 0s - loss: 0.8616 - accuracy: 0.6873
Testing loss: 0.7582, test accuracy: 0.7310
336/336 [==============================] - 77s 230ms/step - loss: 0.8616 - accuracy: 0.6873 - val_loss: 0.7778 - val_accuracy: 0.7240
Epoch 5/10
336/336 [==============================] - ETA: 0s - loss: 0.6501 - accuracy: 0.7651
Testing loss: 0.5808, test accuracy: 0.8015
336/336 [==============================] - 78s 232ms/step - loss: 0.6501 - accuracy: 0.7651 - val_loss: 0.5876 - val_accuracy: 0.8002
Epoch 6/10
336/336 [==============================] - ETA: 0s - loss: 0.4778 - accuracy: 0.8337
Testing loss: 0.5777, test accuracy: 0.8068
336/336 [==============================] - 77s 228ms/step - loss: 0.4778 - accuracy: 0.8337 - val_loss: 0.5651 - val_accuracy: 0.8188
Epoch 7/10
336/336 [==============================] - ETA: 0s - loss: 0.3822 - accuracy: 0.8698
Testing loss: 0.4515, test accuracy: 0.8649
336/336 [==============================] - 79s 234ms/step - loss: 0.3822 - accuracy: 0.8698 - val_loss: 0.4849 - val_accuracy: 0.8423
Epoch 8/10
336/336 [==============================] - ETA: 0s - loss: 0.3156 - accuracy: 0.8952
Testing loss: 0.4869, test accuracy: 0.8509
336/336 [==============================] - 77s 229ms/step - loss: 0.3156 - accuracy: 0.8952 - val_loss: 0.5065 - val_accuracy: 0.8423
Epoch 9/10
336/336 [==============================] - ETA: 0s - loss: 0.2606 - accuracy: 0.9142
Testing loss: 0.5063, test accuracy: 0.8518
336/336 [==============================] - 76s 227ms/step - loss: 0.2606 - accuracy: 0.9142 - val_loss: 0.5044 - val_accuracy: 0.8571
Epoch 10/10
336/336 [==============================] - ETA: 0s - loss: 0.2453 - accuracy: 0.9208
Testing loss: 0.4287, test accuracy: 0.8744
336/336 [==============================] - 78s 233ms/step - loss: 0.2453 - accuracy: 0.9208 - val_loss: 0.4223 - val_accuracy: 0.8761
105/105 [==============================] - 4s 42ms/step - loss: 0.4287 - accuracy: 0.8744
Test accuracy: 87.44%
```

25 Results of Task3 on ten epochs before augmentation

We can find that the training loss starts at a high value and consistently decreases over the epochs, which indicates that the model is learning and improving its performance on the training dataset. The curve is smooth, suggesting a steady learning process without significant disruptions.

The validation loss decreases alongside the training loss, which is a positive sign that the model is generalizing well and not overfitting to the training data. However, the validation loss curve flattens towards the end, which could suggest that the model is beginning to plateau in its learning and might not show significant improvement without further training or adjustments.

The training accuracy starts at a lower value and increases over the epochs, which is expected as the model's loss decreases. The curve's ascent slows down as it progresses, implying that the model is approaching its maximum performance on the training set. The validation accuracy increases almost at the same rate as the training accuracy, reinforcing the idea that the model is generalizing well. However, just like the loss, the accuracy also starts to plateau, which indicates a similar trend of the model reaching its performance limit without further optimizations. The testing accuracy improves significantly over the epochs, showing that the model's generalization is effective. The consistent upward trend without leveling off suggests that the model might still benefit from further training epochs or more complex modeling to achieve higher accuracy. We can see that by the 10th epoch, the model achieves a test accuracy of 87.4%, which is quite high and confirms the trend observed in the graphical representations.

The results suggest that there is a balance between complexity and accuracy that is working well for your current dataset and problem. Since the testing accuracy is still increasing, more training might yield better results, as we could see above 20 epochs gave higher test accuracy than ten epochs. Using data augmentation can help prevent overfitting and improve the model's generalization. We are showing results after augmentation below.

26 Task3 on 20 epochs after augmentation

```
Epoch 13/20
336/336 [==============================] - ETA: 0s - loss: 0.3824 - accuracy: 0.8773
Testing loss: 0.4523, test accuracy: 0.8574
336/336 [==============================] - 77s 229ms/step - loss: 0.3824 - accuracy: 0.8773 - val_loss: 0.5227 -
val_accuracy: 0.8464
Epoch 14/20
336/336 [==============================] - ETA: 0s - loss: 0.3701 - accuracy: 0.8829
Testing loss: 0.3093, test accuracy: 0.9036
336/336 [==============================] - 78s 231ms/step - loss: 0.3701 - accuracy: 0.8829 - val_loss: 0.3012 -
val_accuracy: 0.9081
Epoch 15/20
336/336 [==============================] - ETA: 0s - loss: 0.3607 - accuracy: 0.8856
Testing loss: 0.3354, test accuracy: 0.8988
336/336 [==============================] - 77s 228ms/step - loss: 0.3607 - accuracy: 0.8856 - val_loss: 0.3375 -
val_accuracy: 0.9010
Epoch 16/20
336/336 [==============================] - ETA: 0s - loss: 0.3526 - accuracy: 0.8904
Testing loss: 0.3156, test accuracy: 0.9015
336/336 [==============================] - 78s 233ms/step - loss: 0.3526 - accuracy: 0.8904 - val_loss: 0.3018 -
val_accuracy: 0.9096
Epoch 17/20
336/336 [==============================] - ETA: 0s - loss: 0.3563 - accuracy: 0.8838
Testing loss: 0.3108, test accuracy: 0.9083
336/336 [==============================] - 80s 239ms/step - loss: 0.3563 - accuracy: 0.8838 - val_loss: 0.3294 -
val_accuracy: 0.9059
Epoch 18/20
336/336 [==============================] - ETA: 0s - loss: 0.3476 - accuracy: 0.8919
Testing loss: 0.3080, test accuracy: 0.9033
336/336 [==============================] - 77s 228ms/step - loss: 0.3476 - accuracy: 0.8919 - val_loss: 0.3210 -
val_accuracy: 0.9062
Epoch 19/20
336/336 [==============================] - ETA: 0s - loss: 0.3204 - accuracy: 0.8992
Testing loss: 0.3176, test accuracy: 0.8991
336/336 [==============================] - 77s 228ms/step - loss: 0.3204 - accuracy: 0.8992 - val_loss: 0.3297 -
val_accuracy: 0.8999
Epoch 20/20
336/336 [==============================] - ETA: 0s - loss: 0.3456 - accuracy: 0.8939
Testing loss: 0.3022, test accuracy: 0.9107
336/336 [==============================] - 79s 234ms/step - loss: 0.3456 - accuracy: 0.8939 - val_loss: 0.3073 -
val_accuracy: 0.9055
105/105 [==============================] - 4s 42ms/step - loss: 0.3022 - accuracy: 0.9107
Test accuracy: 91.07%
```

27 Results on 20 epochs after augmentation

As anticipated, the outcomes post-augmentation demonstrated significant improvement. In fact, augmentation exerted a substantial influence, elevating the accuracy from 87.44% to 91.07%.

Comparison Task3&2&1:

Complexity vs. Performance:

Model 1 is the simplest and still achieves high accuracy, making it efficient in terms of computational resources. Model 2, despite its complexity, shows the highest accuracy, which may justify the additional complexity. Model 3 has a deep architecture, which seems to overfit slightly but is controlled with early stopping. Model 4 has lower accuracy compared to the others but still shows a good learning trend.

Accuracy:

Model 2 is the leader in accuracy, both before and after augmentation. Model 1 and Model 3 show competitive accuracy, with Model 3 slightly higher. Model 4 trails with the lowest accuracy but may still have room for improvement.

Generalization:

Models 1 and 2 benefit from data augmentation, indicating that this technique effectively improves their generalization. Model 3's use of early stopping suggests that it might be the most prone to overfitting among the four. Model 4's performance suggests it is generalizing well, but the plateau indicates potential limitations in its current architecture or learning capacity.

Efficiency:

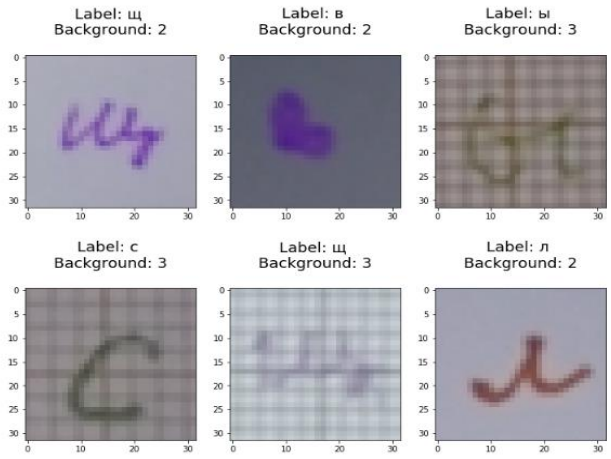
Model 1 stands out for its balance between simplicity and high accuracy. Model 2, while the most accurate, may require more computational power and time due to its complexity. Model 3 and Model 4 might need further optimization for efficiency.

Task 4: Implementing Transfer Learning with Pre-Trained Networks

Transfer learning is a machine learning approach that uses knowledge gained from one task to enhance performance on a related task. Instead of training models from scratch for each specific task, transfer learning allows models to benefit from pre-existing knowledge acquired during training on a source task. By fine-tuning or adapting the pre-trained model on a target task with limited data, transfer learning efficiently utilizes learned features, leading to improved performance and faster convergence. This approach is particularly effective in domains like natural language processing, computer vision, and speech recognition, where obtaining large labeled datasets for specific tasks can be challenging or costly.

Pretrained model and dataset:

The dataset is a collection of hand-written Russian lowercase letters, created from the author's own photos. It includes three main sets: "letters.zip" with 1650 color images, "letters2.zip" with 5940 images, and "letters3.zip" with 6600 images, each containing 33 letters. The images are in PNG format, and corresponding labels are provided in letters.txt, letters2.txt, and letters3.txt. Additionally, there are preprocessing files in the form of LetterColorImages.h5, LetterColorImages2.h5, and LetterColorImages3.h5.



The dataset is freely available on GitHub and is suitable for tasks like classification and image generation involving handwritten letters. The owner encourages improvements, suggesting the

addition of images written by different individuals or the inclusion of capital letters for further enhancement.

The pretrained model for classifying handwritten Russian letters is a convolutional neural network (CNN) with multiple layers.

The architecture begins with a Conv2D layer with a filter size of 32, producing an output shape of (30, 30, 32) and 896 parameters. Subsequently, two more Conv2D layers follow, with filter sizes of 64 and 128, resulting in output shapes of (28, 28, 64) and (25, 25, 128), respectively. A MaxPooling2D layer with a pool size of (2, 2) reduces the spatial dimensions to (12, 12, 128), and a Dropout layer with a dropout rate of 0.5 is applied to prevent overfitting.

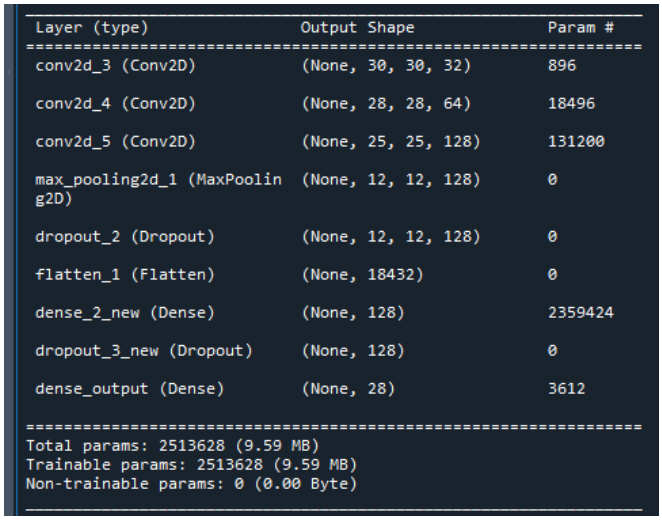
The model then flattens the output to a vector of length 18432, followed by a Dense layer with 128 neurons. Another Dropout layer is employed before the final Dense layer with 33 neurons, representing the number of unique letters in the dataset.

The model is well-suited for handwritten letter classification and has been pretrained on the provided dataset, showcasing its capability to recognize distinct features and patterns in the images.

Fine tuning:

(New model)

```
# Load your pre-trained model
# Define custom metric if it's not a standard one
def top_3_categorical_accuracy(y_true, y_pred):
    return TopKCategoricalAccuracy(k=3)(y_true, y_pred)

# Load your pre-trained model with custom objects
pretrained_model = load_model('/kaggle/input/modeltask4/model (1).h5',
                              custom_objects={'top_3_categorical_accuracy': top_3_categorical_accuracy})

# Create a new Sequential model
model = Sequential()

# Add layers from the pre-trained model up to the desired point
for layer in pretrained_model.layers[:-3]:  # Exclude the last 3 layers (flatten, dense_2, dropout_3)
    model.add(layer)

# Add new layers for the new task
model.add(Dense(128, activation='relu', name='dense_2_new'))
model.add(Dropout(0.5, name='dropout_3_new'))

# Change output layer for new classification task
model.add(Dense(28, activation='softmax', name='dense_output'))

# Print the modified model summary
model.summary()
# Compile the model with Adam optimizer
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
```



Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 30, 30, 32)	896
conv2d_4 (Conv2D)	(None, 28, 28, 64)	18496
conv2d_5 (Conv2D)	(None, 25, 25, 128)	131200
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 128)	0
dropout_2 (Dropout)	(None, 12, 12, 128)	0
flatten_1 (Flatten)	(None, 18432)	0
dense_2_new (Dense)	(None, 128)	2359424
dropout_3_new (Dropout)	(None, 128)	0
dense_output (Dense)	(None, 28)	3612

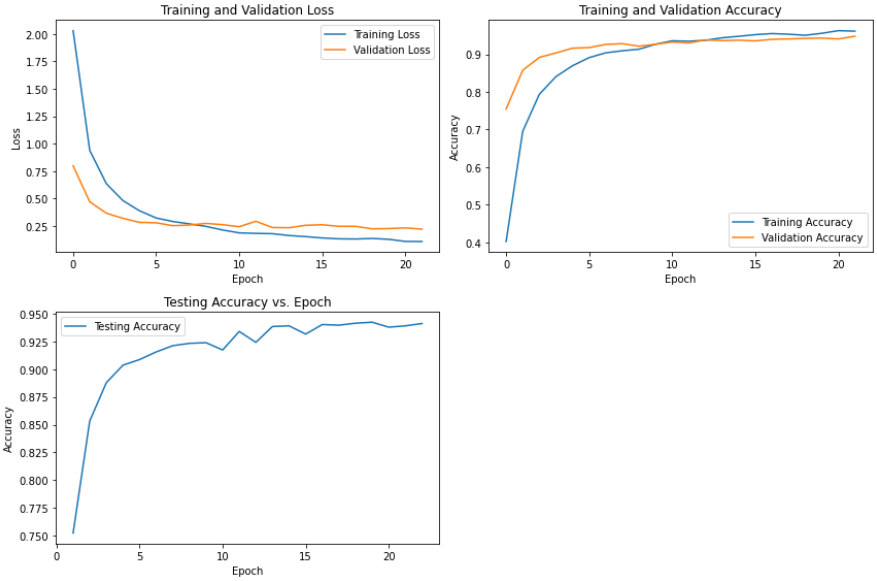
Total params: 2513628 (9.59 MB)
Trainable params: 2513628 (9.59 MB)
Non-trainable params: 0 (0.00 Byte)

In this adaptation, a pre-trained convolutional neural network (CNN) model originally designed for classifying handwritten Russian letters was repurposed to classify handwritten Arabic letters. The model, stored as an HDF5 file ('model.h5'), underwent modifications to accommodate the unique characteristics of Arabic script, such as different shapes and characters compared to Russian letters.

The input layer of the original model was adjusted to match the new input shape of (32, 32, 1) suitable for handwritten Arabic letters. The output layer was replaced with a new dense layer containing 28 neurons, aligning with the number of classes corresponding to Arabic letters, and a softmax activation function for multi-class classification. Intermediate layers, 'dense_2_new' and 'dropout_3_new', were introduced to facilitate the model's adaptation to the intricacies of Arabic handwriting.

The transfer learning approach leveraged the knowledge and feature extraction capabilities acquired by the original model during the training on Russian letters. By fine-tuning the model on the new dataset of handwritten Arabic letters, the modified model aimed to capture the relevant features for accurate classification. This process illustrates the versatility of transfer learning, allowing the reuse of pre-trained models across different character sets and languages, with adjustments made to suit the specific requirements of the new classification task.

Results:




```
val_accuracy: 0.9408
Epoch 19/22
336/336 [==============================] - ETA: 0s - loss: 0.1366 - accuracy: 0.9506
Testing loss: 0.2423, test accuracy: 0.9423
336/336 [==============================] - 23s 69ms/step - loss: 0.1366 - accuracy: 0.9506 - val_loss: 0.2245 -
val_accuracy: 0.9423
Epoch 20/22
336/336 [==============================] - ETA: 0s - loss: 0.1290 - accuracy: 0.9558
Testing loss: 0.2639, test accuracy: 0.9378
336/336 [==============================] - 24s 72ms/step - loss: 0.1290 - accuracy: 0.9558 - val_loss: 0.2269 -
val_accuracy: 0.9431
Epoch 21/22
336/336 [==============================] - ETA: 0s - loss: 0.1096 - accuracy: 0.9629
Testing loss: 0.2584, test accuracy: 0.9390
336/336 [==============================] - 24s 70ms/step - loss: 0.1096 - accuracy: 0.9629 - val_loss: 0.2328 -
val_accuracy: 0.9408
Epoch 22/22
336/336 [==============================] - ETA: 0s - loss: 0.1091 - accuracy: 0.9615
Testing loss: 0.2527, test accuracy: 0.9411
336/336 [==============================] - 23s 69ms/step - loss: 0.1091 - accuracy: 0.9615 - val_loss: 0.2217 -
val_accuracy: 0.9479
105/105 [==============================] - 2s 15ms/step - loss: 0.2527 - accuracy: 0.9411
Test accuracy: 94.11%
```

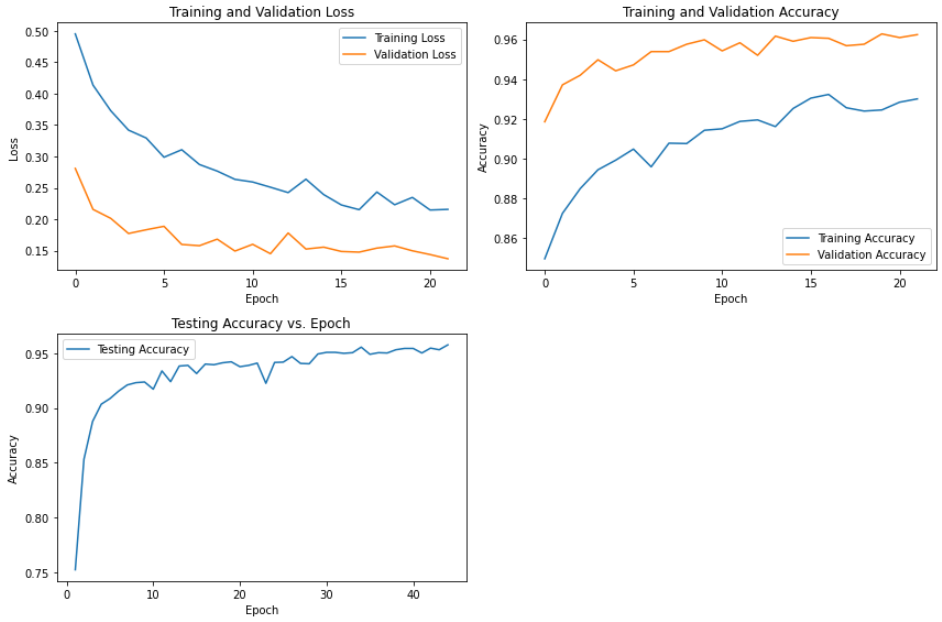
We trained the model for 22 epochs, achieving a test accuracy of approximately 94.1%. The model demonstrated effective learning, as evidenced by the rapidly decreasing training loss, which plateaued at a low value, indicating successful predictions of the correct labels for training examples. Likewise, the validation loss decreased and stabilized, which signifies the model's robust generalization to unseen data.

Throughout the training process, we observed an increase in both training and validation accuracy, with training accuracy reaching about 95.6% and validation accuracy peaking around 94.8%. This shows the model's improved ability to correctly classify both training examples and new instances from the validation set.

These outcomes are indicative of the model's effective learning and its generalization capabilities. Nonetheless, it's crucial to recognize that after 22 epochs, there might be potential for further enhancements with additional training epochs.

It is also important to note that while the validation accuracy is high, it is not at 100%, which means there are some errors in the model's predictions. However, the relatively high validation accuracy implies that the model is making few mistakes, which affirms its efficacy in the given classification tasks.

Augmentation:




```
Epoch 17/22
335/336 [============================>.] - ETA: 0s - loss: 0.2147 - accuracy: 0.9325
Testing loss: 0.1721, test accuracy: 0.9545
336/336 [==============================] - 17s 49ms/step - loss: 0.2152 - accuracy: 0.9323 - val_loss: 0.1476 -
val_accuracy: 0.9606
Epoch 18/22
336/336 [==============================] - ETA: 0s - loss: 0.2434 - accuracy: 0.9256
Testing loss: 0.1724, test accuracy: 0.9545
336/336 [==============================] - 16s 47ms/step - loss: 0.2434 - accuracy: 0.9256 - val_loss: 0.1540 -
val_accuracy: 0.9568
Epoch 19/22
335/336 [============================>.] - ETA: 0s - loss: 0.2222 - accuracy: 0.9241
Testing loss: 0.1773, test accuracy: 0.9503
336/336 [==============================] - 15s 44ms/step - loss: 0.2231 - accuracy: 0.9239 - val_loss: 0.1575 -
val_accuracy: 0.9576
Epoch 20/22
336/336 [==============================] - ETA: 0s - loss: 0.2348 - accuracy: 0.9245
Testing loss: 0.1719, test accuracy: 0.9548
336/336 [==============================] - 16s 46ms/step - loss: 0.2348 - accuracy: 0.9245 - val_loss: 0.1497 -
val_accuracy: 0.9628
Epoch 21/22
335/336 [============================>.] - ETA: 0s - loss: 0.2152 - accuracy: 0.9282
Testing loss: 0.1724, test accuracy: 0.9533
336/336 [==============================] - 16s 47ms/step - loss: 0.2147 - accuracy: 0.9284 - val_loss: 0.1437 -
val_accuracy: 0.9609
Epoch 22/22
336/336 [==============================] - ETA: 0s - loss: 0.2157 - accuracy: 0.9301
Testing loss: 0.1643, test accuracy: 0.9577
336/336 [==============================] - 15s 45ms/step - loss: 0.2157 - accuracy: 0.9301 - val_loss: 0.1370 -
val_accuracy: 0.9624
105/105 [==============================] - 2s 14ms/step - loss: 0.1643 - accuracy: 0.9577
Test accuracy: 95.77%
```

Following the data augmentation, the retrained CNN model exhibits notable improvement in classifying Arabic letters, with a test accuracy reaching 95.77%. The training and validation loss both show a decreasing trend, with the training loss stabilizing at a lower level than the validation loss. This demonstrates the model's strong learning capabilities without overfitting, as the validation loss remains close to the training loss.

The training accuracy displays an upward trend, achieving an impressive 95.77%, while the validation accuracy also increases, peaking at 96.09%. The close alignment between the training and validation accuracy suggests that the model is generalizing well to unseen data. There is no pronounced gap between the training and validation accuracy, indicating that the model is not overfitting, which is a common concern in machine learning models.

Overall, the post-augmentation results are highly encouraging, showing that the model has successfully adapted from recognizing Russian letters to Arabic letters with high accuracy. The training process appears to be stable, and the model demonstrates good generalization. While there are minor fluctuations in validation accuracy, they do not detract significantly from the model's performance. Further fine-tuning, if necessary, could potentially make the model even more robust to variations in the data.

Final comparison

1. **Model 1: Efficient Balance**
    - Model 1, with its simpler architecture, achieved a test accuracy of about 91.4% before augmentation and slightly higher post-augmentation. Its strength lies in its efficiency and balance between simplicity and accuracy.
2. **Model 2: Highest Accuracy**
    - Model 2, being more complex, had a test accuracy of 95.12% before augmentation and increased to 96.22% after augmentation. This model is the leader in terms of accuracy, although it demands more computational resources.
3. **Model 3: Deep and Controlled**
    - With its deep architecture, Model 3 achieved a test accuracy of 93.54%. The use of early stopping to prevent overfitting suggests a controlled approach, making it a good option for tasks requiring depth and nuanced understanding.
4. **Model 4: (Task3)**
    - Model 4 shows a good learning trend with a test accuracy of 87.4% at 10 epochs and 91.07% after augmentation at 20 epochs. This model has potential for further improvement with additional training or optimizations.
5. **Transfer Learning Model: Robust and Adaptable**
    - The updated results show the transfer learning model achieving a test accuracy of 94.1%, which further improves to 95.77% post-augmentation. This performance

is quite remarkable, considering it has been adapted from a model originally trained on Russian letters.

- The training and validation accuracy are impressively close, with the training accuracy at around 95.6% and validation accuracy peaking at 96.09% post-augmentation. This close alignment indicates effective generalization and minimal overfitting.

- The efficiency of this model, in terms of computational resources and its adaptability to new tasks, makes it a highly valuable option, especially when dealing with related but distinct datasets.

4. Conclusion

In conclusion, this project embarked on a comprehensive exploration of various convolutional neural network (CNN) models, assessing their performance across multiple tasks. Through rigorous testing and comparison, we identified crucial trade-offs between model complexity and performance, the efficacy of simpler models in certain contexts, and the potential of deep architectures when properly managed. Significantly, the project underscored the value of transfer learning in enhancing model adaptability and efficiency, showcasing its power in leveraging pre-existing models for new tasks with remarkable success.

Our findings highlight the importance of strategic model selection, attentive to the balance between computational demands and desired accuracy. The project also reaffirmed the iterative nature of model optimization, where continuous refinement and adjustments lead to improved outcomes. In essence, our work not only contributes to the broader understanding of CNN capabilities and limitations but also sets a foundation for future projects aiming to harness deep learning technologies efficiently and effectively.

References:

1) Computer vision slides, 24 January 2024 . [Online].