Generative AI (CNN, RNN , PixelCNN, RowLSTM, Diagonal BiLSTM)

M Abdurrahman Khan[1]

Fast University i221148@nu.edu.pk

Abstract. This work reports three experiments: (1) convolutional neural networks (CNNs) for image classification on CIFAR-10, (2) LSTMbased models for text generation on Shakespeare's dataset, and (3) PixelRNN (PixelCNN, Row LSTM, Diagonal BiLSTM) for image modeling. We analyze the impact of depth, hyperparameters, and regularization in CNNs, evaluate hidden units and dropout in LSTMs, and compare autoregressive models using bits per dimension. The results show that architecture design and hyperparameter choices are critical across vision and language tasks.

Keywords: CNN · LSTM · CIFAR-10 · Text Generation · PixelCNN · PixelRNN

1 Question 1: CNN For CIFAR-10

1.1 Introduction

The CIFAR-10 dataset is a benchmark for image classification. This study explores CNN architectures with 3, 5, and 7 convolutional layers, trained with different batch sizes and learning rates. We report performance metrics such as accuracy, precision, recall, and F1-score to analyze model behavior.

1.2 Methodology

We implemented a CNN in TensorFlow/Keras with convolution, batch normalization, max pooling, dropout, and dense layers. Hyperparameters explored include:

- Learning rates: 0.001, 0.01, 0.1
- Batch sizes: 16, 32, 64
- Convolutional filters: (16,32,64), (32,64,128), up to (16,32,64,128,256,512,512)
- Number of CNN layers: 3, 5, and 7

1.3 Results

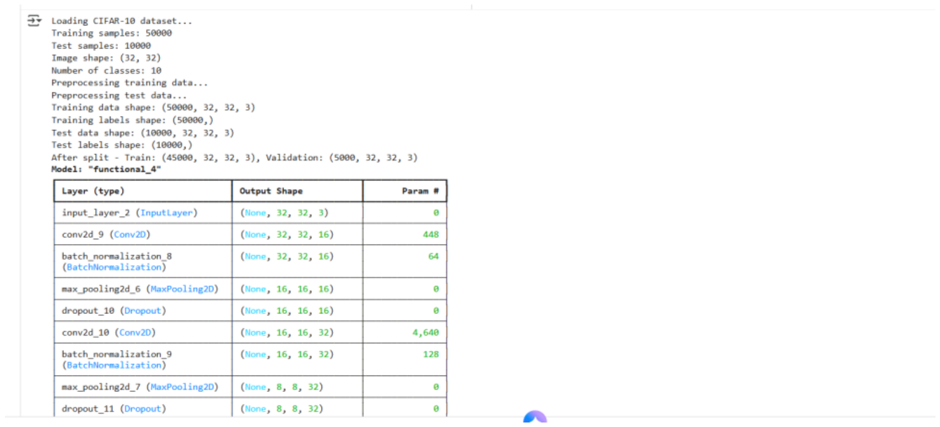
Table 1 summarizes the results of CNN models under different configurations.

Table 1. CNN Results on CIFAR-10 with different hyperparameters

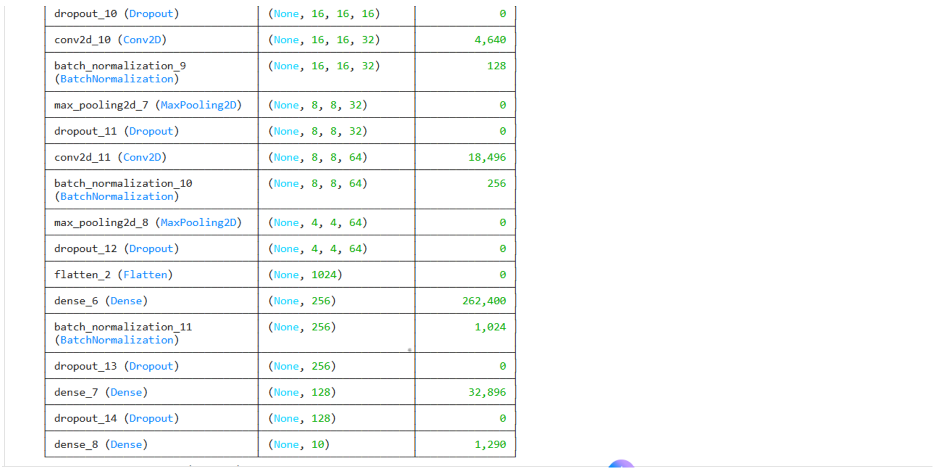
Model	LR	Batch	Conv Filters	Layers	Acc.	Prec.	Recall	F1
CNN	0.001	16	16,32,64	3	0.7651	0.7639	0.7651	0.7624
CNN	0.01	16	16,32,64	3	0.5296	0.5174	0.5296	0.5010
CNN	0.1	16	16,32,64	3	0.1003	0.0700	0.1003	0.0188
CNN	0.001	32	16,32,64	3	0.7756	0.7796	0.7756	0.7711
CNN	0.001	64	16,32,64	3	0.7675	0.7707	0.7675	0.7647
CNN	0.001	64	32,64,128	3	0.8096	0.8094	0.8096	0.8089
CNN	0.001	64	16,32,64,128,256	5	0.7911	0.7917	0.7911	0.7893
CNN	0.001	64	16,32,64,128,256,512,512	7	0.8592	0.8583	0.8592	0.8584

1.4 Discussion

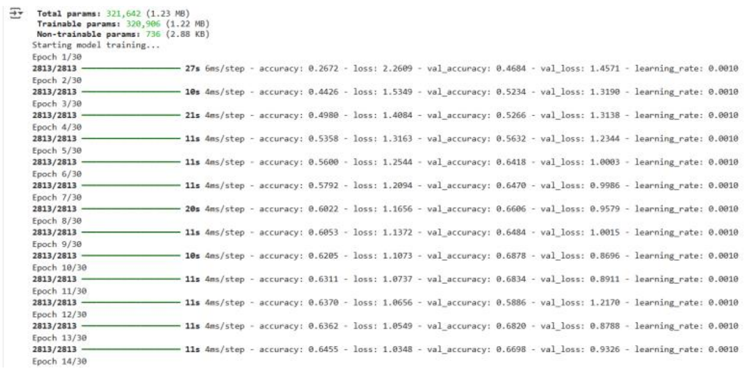
- Higher learning rates (0.1) lead to unstable training and poor results.
- Increasing batch size to 32 or 64 improves stability and generalization.
- Deeper networks (7 layers) achieve the best accuracy (85.92%).
- Filter scaling from (16,32,64) to (32,64,128) yields significant performance gains.



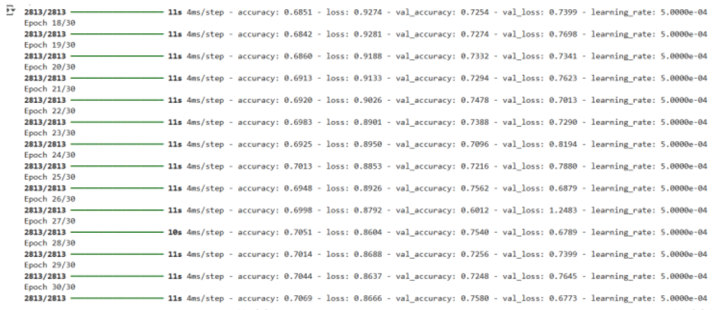
```
Loading CIFAR-10 dataset...
Training samples: 50000
Test samples: 10000
Image shape: (32, 32)
Number of classes: 10
Preprocessing training data...
Preprocessing test data...
Training data shape: (50000, 32, 32, 3)
Training labels shape: (50000,)
Test data shape: (10000, 32, 32, 3)
Test labels shape: (10000,)
After split - Train: (45000, 32, 32, 3), Validation: (5000, 32, 32, 3)
Model: "functional_4"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━┓
┃ Layer (type)                        ┃ Output Shape           ┃    Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━┩
│ input_layer_2 (InputLayer)          │ (None, 32, 32, 3)      │          0 │
├─────────────────────────────────────┼────────────────────────┼────────────┤
│ conv2d_9 (Conv2D)                   │ (None, 32, 32, 16)     │        448 │
├─────────────────────────────────────┼────────────────────────┼────────────┤
│ batch_normalization_8               │ (None, 32, 32, 16)     │         64 │
│ (BatchNormalization)                │                        │            │
├─────────────────────────────────────┼────────────────────────┼────────────┤
│ max_pooling2d_6 (MaxPooling2D)      │ (None, 16, 16, 16)     │          0 │
├─────────────────────────────────────┼────────────────────────┼────────────┤
│ dropout_10 (Dropout)                │ (None, 16, 16, 16)     │          0 │
├─────────────────────────────────────┼────────────────────────┼────────────┤
│ conv2d_10 (Conv2D)                  │ (None, 16, 16, 32)     │      4,640 │
├─────────────────────────────────────┼────────────────────────┼────────────┤
│ batch_normalization_9               │ (None, 16, 16, 32)     │        128 │
│ (BatchNormalization)                │                        │            │
├─────────────────────────────────────┼────────────────────────┼────────────┤
│ max_pooling2d_7 (MaxPooling2D)      │ (None, 8, 8, 32)       │          0 │
├─────────────────────────────────────┼────────────────────────┼────────────┤
│ dropout_11 (Dropout)                │ (None, 8, 8, 32)       │          0 │
```



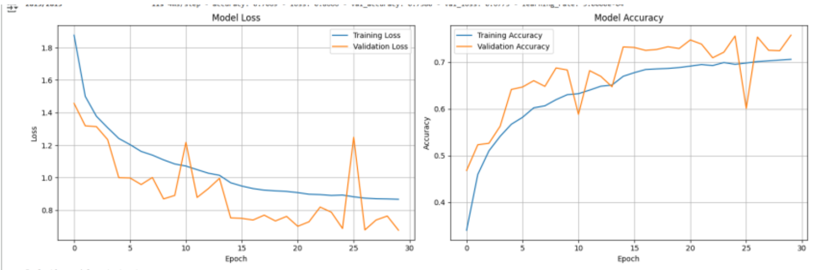
dropout_10 (Dropout)	(None, 16, 16, 16)	0
conv2d_10 (Conv2D)	(None, 16, 16, 32)	4,640
batch_normalization_9 (BatchNormalization)	(None, 16, 16, 32)	128
max_pooling2d_7 (MaxPooling2D)	(None, 8, 8, 32)	0
dropout_11 (Dropout)	(None, 8, 8, 32)	0
conv2d_11 (Conv2D)	(None, 8, 8, 64)	18,496
batch_normalization_10 (BatchNormalization)	(None, 8, 8, 64)	256
max_pooling2d_8 (MaxPooling2D)	(None, 4, 4, 64)	0
dropout_12 (Dropout)	(None, 4, 4, 64)	0
flatten_2 (Flatten)	(None, 1024)	0
dense_6 (Dense)	(None, 256)	262,400
batch_normalization_11 (BatchNormalization)	(None, 256)	1,024
dropout_13 (Dropout)	(None, 256)	0
dense_7 (Dense)	(None, 128)	32,896
dropout_14 (Dropout)	(None, 128)	0
dense_8 (Dense)	(None, 10)	1,290



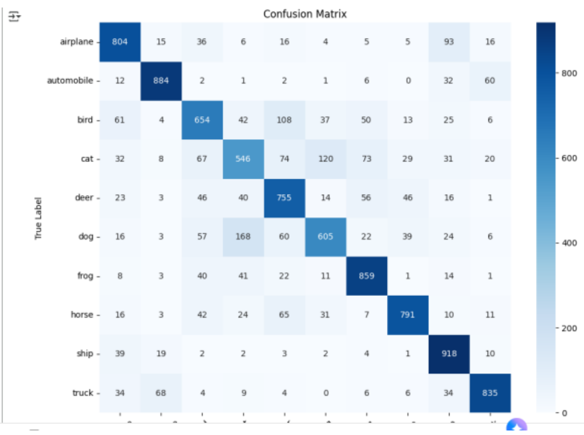
```
 Total params: 321,642 (1.23 MB)
 Trainable params: 320,906 (1.22 MB)
 Non-trainable params: 736 (2.88 KB)
Starting model training...
Epoch 1/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 27s 6ms/step - accuracy: 0.2672 - loss: 2.2609 - val_accuracy: 0.4684 - val_loss: 1.4571 - learning_rate: 0.0010
Epoch 2/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.4426 - loss: 1.5349 - val_accuracy: 0.5234 - val_loss: 1.3190 - learning_rate: 0.0010
Epoch 3/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 21s 4ms/step - accuracy: 0.4980 - loss: 1.4084 - val_accuracy: 0.5266 - val_loss: 1.3138 - learning_rate: 0.0010
Epoch 4/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.5358 - loss: 1.3163 - val_accuracy: 0.5632 - val_loss: 1.2344 - learning_rate: 0.0010
Epoch 5/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.5600 - loss: 1.2544 - val_accuracy: 0.6418 - val_loss: 1.0003 - learning_rate: 0.0010
Epoch 6/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.5792 - loss: 1.2094 - val_accuracy: 0.6470 - val_loss: 0.9986 - learning_rate: 0.0010
Epoch 7/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 20s 4ms/step - accuracy: 0.6022 - loss: 1.1656 - val_accuracy: 0.6606 - val_loss: 0.9579 - learning_rate: 0.0010
Epoch 8/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.6053 - loss: 1.1372 - val_accuracy: 0.6484 - val_loss: 1.0015 - learning_rate: 0.0010
Epoch 9/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.6205 - loss: 1.1073 - val_accuracy: 0.6878 - val_loss: 0.8696 - learning_rate: 0.0010
Epoch 10/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.6311 - loss: 1.0737 - val_accuracy: 0.6834 - val_loss: 0.8911 - learning_rate: 0.0010
Epoch 11/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.6370 - loss: 1.0656 - val_accuracy: 0.5886 - val_loss: 1.2170 - learning_rate: 0.0010
Epoch 12/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.6362 - loss: 1.0549 - val_accuracy: 0.6820 - val_loss: 0.8788 - learning_rate: 0.0010
Epoch 13/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.6455 - loss: 1.0348 - val_accuracy: 0.6698 - val_loss: 0.9326 - learning_rate: 0.0010
Epoch 14/30
```



```
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.6851 - loss: 0.9274 - val_accuracy: 0.7254 - val_loss: 0.7399 - learning_rate: 5.0000e-04
Epoch 18/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.6842 - loss: 0.9281 - val_accuracy: 0.7274 - val_loss: 0.7698 - learning_rate: 5.0000e-04
Epoch 19/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.6860 - loss: 0.9188 - val_accuracy: 0.7332 - val_loss: 0.7341 - learning_rate: 5.0000e-04
Epoch 20/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.6913 - loss: 0.9133 - val_accuracy: 0.7294 - val_loss: 0.7623 - learning_rate: 5.0000e-04
Epoch 21/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.6920 - loss: 0.9026 - val_accuracy: 0.7478 - val_loss: 0.7013 - learning_rate: 5.0000e-04
Epoch 22/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.6983 - loss: 0.8901 - val_accuracy: 0.7388 - val_loss: 0.7290 - learning_rate: 5.0000e-04
Epoch 23/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.6925 - loss: 0.8950 - val_accuracy: 0.7096 - val_loss: 0.8194 - learning_rate: 5.0000e-04
Epoch 24/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.7013 - loss: 0.8853 - val_accuracy: 0.7216 - val_loss: 0.7880 - learning_rate: 5.0000e-04
Epoch 25/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.6948 - loss: 0.8926 - val_accuracy: 0.7562 - val_loss: 0.6879 - learning_rate: 5.0000e-04
Epoch 26/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.6998 - loss: 0.8792 - val_accuracy: 0.6012 - val_loss: 1.2483 - learning_rate: 5.0000e-04
Epoch 27/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.7051 - loss: 0.8604 - val_accuracy: 0.7540 - val_loss: 0.6789 - learning_rate: 5.0000e-04
Epoch 28/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.7014 - loss: 0.8688 - val_accuracy: 0.7256 - val_loss: 0.7399 - learning_rate: 5.0000e-04
Epoch 29/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.7044 - loss: 0.8637 - val_accuracy: 0.7248 - val_loss: 0.7645 - learning_rate: 5.0000e-04
Epoch 30/30
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.7069 - loss: 0.8666 - val_accuracy: 0.7580 - val_loss: 0.6773 - learning_rate: 5.0000e-04
```



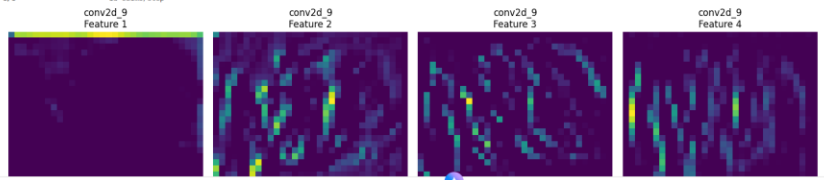
```
Evaluating model on test set...
Test Loss: 0.6764
Test Accuracy: 0.7651
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
Generating confusion matrix...
```

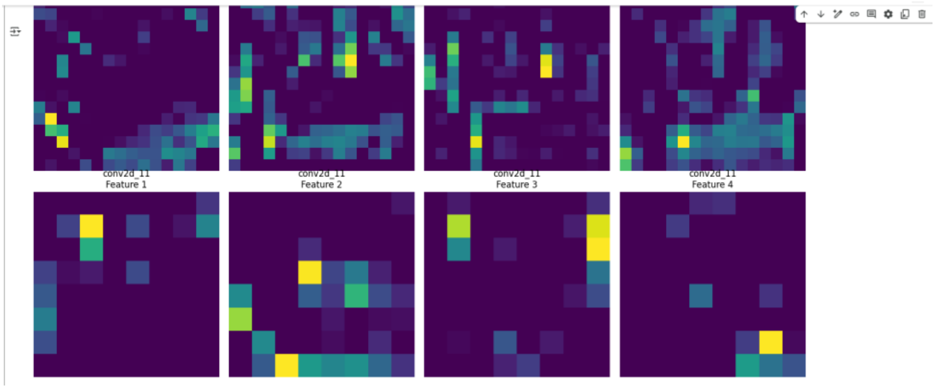


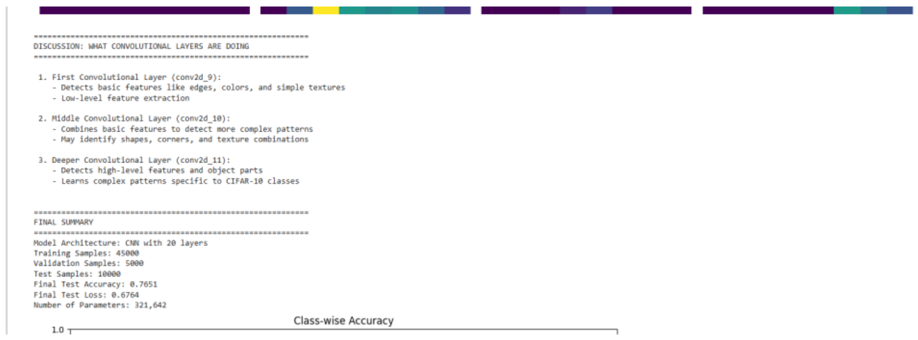
airplane	804	15	36	6	16	4	5	5	93	16
automobile	12	884	2	1	2	1	6	0	32	60
bird	61	4	654	42	108	37	50	13	25	6
cat	32	8	67	546	74	120	73	29	31	20
deer	23	3	46	40	755	14	56	46	16	1
dog	16	3	57	168	60	605	22	39	24	6
frog	8	3	40	41	22	11	859	1	14	1
horse	16	3	42	24	65	31	7	791	10	11
ship	39	19	2	2	3	2	4	1	918	10
truck	34	68	4	9	4	0	6	6	34	835

Predicted Label

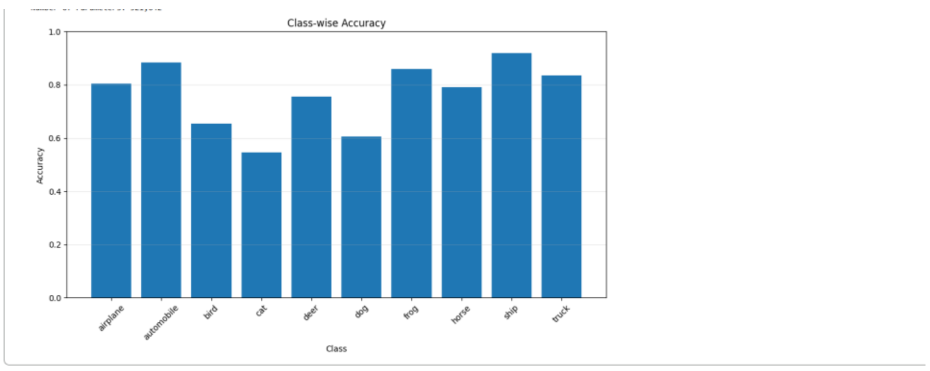
```
Calculating performance metrics...

============================================================
PERFORMANCE METRICS
============================================================
         Class  Precision  Recall  F1-Score  Support
0     airplane     0.7694  0.8040    0.7863     1000
1   automobile     0.8752  0.8840    0.8796     1000
2         bird     0.6884  0.6540    0.6708     1000
3          cat     0.6212  0.5460    0.5812     1000
4         deer     0.6808  0.7550    0.7160     1000
5          dog     0.7333  0.6050    0.6630     1000
6         frog     0.7895  0.8590    0.8228     1000
7        horse     0.8496  0.7910    0.8193     1000
8         ship     0.7669  0.9180    0.8357     1000
9        truck     0.8644  0.8350    0.8494     1000
10     OVERALL     0.7639  0.7651    0.7624    10000
Conv layers found: ['conv2d_9', 'conv2d_10', 'conv2d_11']
Visualizing feature maps...
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 681ms/step
```







```
================================================================
DISCUSSION: WHAT CONVOLUTIONAL LAYERS ARE DOING
================================================================

 1. First Convolutional Layer (conv2d_9):
    - Detects basic features like edges, colors, and simple textures
    - Low-level feature extraction

 2. Middle Convolutional Layer (conv2d_10):
    - Combines basic features to detect more complex patterns
    - May identify shapes, corners, and texture combinations

 3. Deeper Convolutional Layer (conv2d_11):
    - Detects high-level features and object parts
    - Learns complex patterns specific to CIFAR-10 classes


================================================================
FINAL SUMMARY
================================================================
Model Architecture: CNN with 20 layers
Training Samples: 45000
Validation Samples: 5000
Test Samples: 10000
Final Test Accuracy: 0.7651
Final Test Loss: 0.6764
Number of Parameters: 321,642
```



2 Question 2: RNN For Next Word Prediction

2.1 Introduction

The second experiment applies recurrent neural networks to the Tiny Shakespeare dataset. We tokenize text at the word level with a limited vocabulary, train BiLSTM and LSTM layers with dropout, and evaluate models using accuracy and perplexity.

2.2 Methodology

The model includes:

– Word-level tokenization with a vocabulary limit of 8000 and an <UNK> token. – Embedding layer (300 dimensions).
– Two stacked BiLSTM layers and one LSTM layer with hidden units = 600. – Dense and dropout layers for regularization.
– Adam optimizer with gradient clipping, early stopping, and learning rate scheduling.

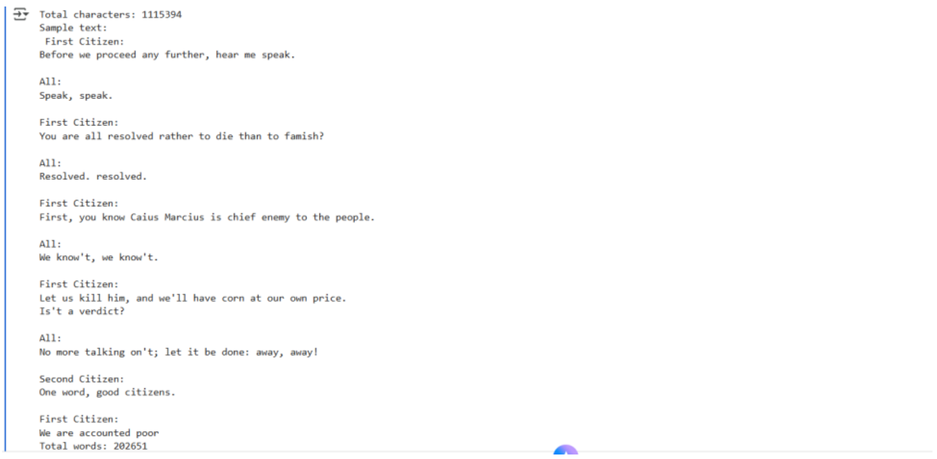
2.3 Results

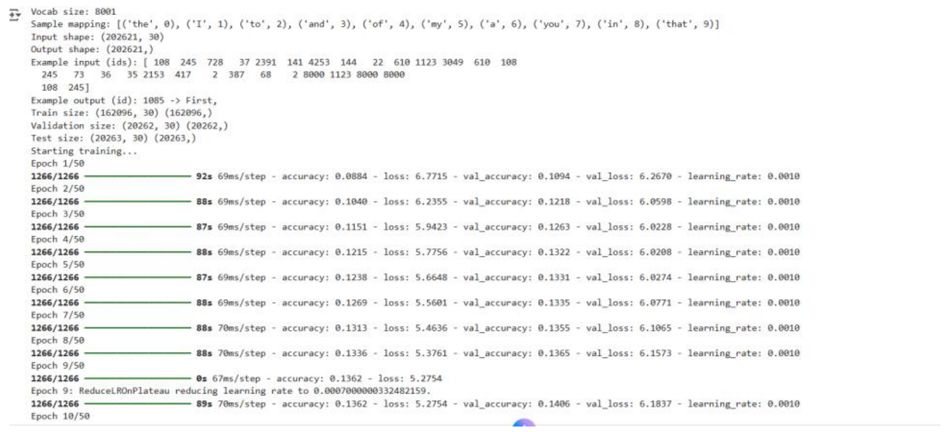
Table 2 summarizes test accuracy, perplexity, and sample generated text.

Table 2. LSTM Results on Tiny Shakespeare with different configurations

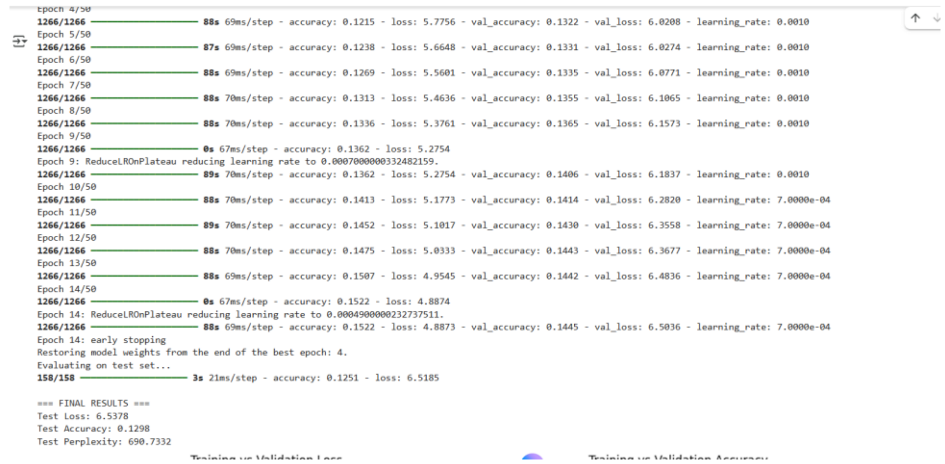
2.4 Discussion

– BiLSTM models capture richer context, improving text quality but not always reducing perplexity.
– Higher dropout (0.3) sometimes degrades accuracy but reduces overfitting.
– Generated text shows grammatical structure but semantic repetition.
– Perplexity values confirm the challenge of word-level text generation on small datasets.

–

```
Total characters: 1115394
Sample text:
 First Citizen:
Before we proceed any further, hear me speak.

All:
Speak, speak.

First Citizen:
You are all resolved rather to die than to famish?

All:
Resolved. resolved.

First Citizen:
First, you know Caius Marcius is chief enemy to the people.

All:
We know't, we know't.

First Citizen:
Let us kill him, and we'll have corn at our own price.
Is't a verdict?

All:
No more talking on't; let it be done: away, away!

Second Citizen:
One word, good citizens.

First Citizen:
We are accounted poor
Total words: 202651
```

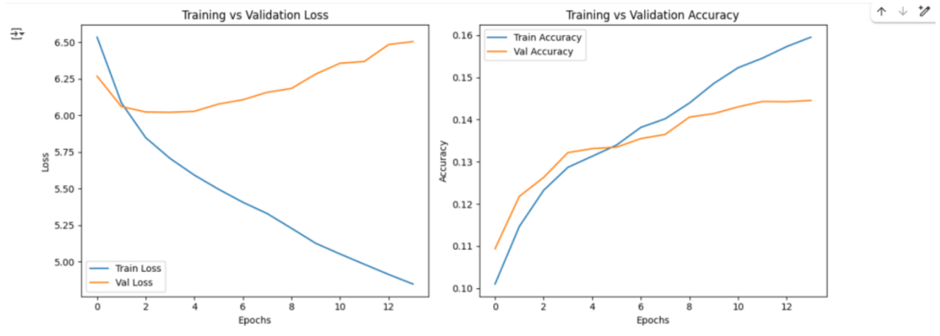


```
Vocab size: 8001
Sample mapping: [('the', 0), ('I', 1), ('to', 2), ('and', 3), ('of', 4), ('my', 5), ('a', 6), ('you', 7), ('in', 8), ('that', 9)]
Input shape: (202621, 30)
Output shape: (202621,)
Example input (ids): [ 108  245  728   37 2391  141 4253  144   22  610 1123 3049  610  108
  245   73   36   35 2153  417    2  387   68    2 8000 1123 8000 8000
  108  245]
Example output (id): 1085 -> First,
Train size: (162096, 30) (162096,)
Validation size: (20262, 30) (20262,)
Test size: (20263, 30) (20263,)
Starting training...
Epoch 1/50
1266/1266 ━━━━━━━━━━━━━━━━━━━━ 92s 69ms/step - accuracy: 0.0884 - loss: 6.7715 - val_accuracy: 0.1094 - val_loss: 6.2670 - learning_rate: 0.0010
Epoch 2/50
1266/1266 ━━━━━━━━━━━━━━━━━━━━ 88s 69ms/step - accuracy: 0.1040 - loss: 6.2355 - val_accuracy: 0.1218 - val_loss: 6.0598 - learning_rate: 0.0010
Epoch 3/50
1266/1266 ━━━━━━━━━━━━━━━━━━━━ 87s 69ms/step - accuracy: 0.1151 - loss: 5.9423 - val_accuracy: 0.1263 - val_loss: 6.0228 - learning_rate: 0.0010
Epoch 4/50
1266/1266 ━━━━━━━━━━━━━━━━━━━━ 88s 69ms/step - accuracy: 0.1215 - loss: 5.7756 - val_accuracy: 0.1322 - val_loss: 6.0208 - learning_rate: 0.0010
Epoch 5/50
1266/1266 ━━━━━━━━━━━━━━━━━━━━ 87s 69ms/step - accuracy: 0.1238 - loss: 5.6648 - val_accuracy: 0.1331 - val_loss: 6.0274 - learning_rate: 0.0010
Epoch 6/50
1266/1266 ━━━━━━━━━━━━━━━━━━━━ 88s 69ms/step - accuracy: 0.1269 - loss: 5.5601 - val_accuracy: 0.1335 - val_loss: 6.0771 - learning_rate: 0.0010
Epoch 7/50
1266/1266 ━━━━━━━━━━━━━━━━━━━━ 88s 70ms/step - accuracy: 0.1313 - loss: 5.4636 - val_accuracy: 0.1355 - val_loss: 6.1065 - learning_rate: 0.0010
Epoch 8/50
1266/1266 ━━━━━━━━━━━━━━━━━━━━ 88s 70ms/step - accuracy: 0.1336 - loss: 5.3761 - val_accuracy: 0.1365 - val_loss: 6.1573 - learning_rate: 0.0010
Epoch 9/50
1266/1266 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step - accuracy: 0.1362 - loss: 5.2754
Epoch 9: ReduceLROnPlateau reducing learning rate to 0.0007000000332482159.
1266/1266 ━━━━━━━━━━━━━━━━━━━━ 89s 70ms/step - accuracy: 0.1362 - loss: 5.2754 - val_accuracy: 0.1406 - val_loss: 6.1837 - learning_rate: 0.0010
Epoch 10/50
```



```
Epoch 4/50
1266/1266 ━━━━━━━━━━━━━━━━━━━━ 88s 69ms/step - accuracy: 0.1215 - loss: 5.7756 - val_accuracy: 0.1322 - val_loss: 6.0208 - learning_rate: 0.0010
Epoch 5/50
1266/1266 ━━━━━━━━━━━━━━━━━━━━ 87s 69ms/step - accuracy: 0.1238 - loss: 5.6648 - val_accuracy: 0.1331 - val_loss: 6.0274 - learning_rate: 0.0010
Epoch 6/50
1266/1266 ━━━━━━━━━━━━━━━━━━━━ 88s 69ms/step - accuracy: 0.1269 - loss: 5.5601 - val_accuracy: 0.1335 - val_loss: 6.0771 - learning_rate: 0.0010
Epoch 7/50
1266/1266 ━━━━━━━━━━━━━━━━━━━━ 88s 70ms/step - accuracy: 0.1313 - loss: 5.4636 - val_accuracy: 0.1355 - val_loss: 6.1065 - learning_rate: 0.0010
Epoch 8/50
1266/1266 ━━━━━━━━━━━━━━━━━━━━ 88s 70ms/step - accuracy: 0.1336 - loss: 5.3761 - val_accuracy: 0.1365 - val_loss: 6.1573 - learning_rate: 0.0010
Epoch 9/50
1266/1266 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step - accuracy: 0.1362 - loss: 5.2754
Epoch 9: ReduceLROnPlateau reducing learning rate to 0.0007000000332482159.
1266/1266 ━━━━━━━━━━━━━━━━━━━━ 89s 70ms/step - accuracy: 0.1362 - loss: 5.2754 - val_accuracy: 0.1406 - val_loss: 6.1837 - learning_rate: 0.0010
Epoch 10/50
```

```
1266/1266 ━━━━━━━━━━━━━━━━━━━━ 88s 70ms/step - accuracy: 0.1413 - loss: 5.1773 - val_accuracy: 0.1414 - val_loss: 6.2820 - learning_rate: 7.0000e-04
Epoch 11/50
1266/1266 ━━━━━━━━━━━━━━━━━━━━ 89s 70ms/step - accuracy: 0.1452 - loss: 5.1017 - val_accuracy: 0.1430 - val_loss: 6.3558 - learning_rate: 7.0000e-04
Epoch 12/50
1266/1266 ━━━━━━━━━━━━━━━━━━━━ 88s 70ms/step - accuracy: 0.1475 - loss: 5.0333 - val_accuracy: 0.1443 - val_loss: 6.3677 - learning_rate: 7.0000e-04
Epoch 13/50
1266/1266 ━━━━━━━━━━━━━━━━━━━━ 88s 69ms/step - accuracy: 0.1507 - loss: 4.9545 - val_accuracy: 0.1442 - val_loss: 6.4836 - learning_rate: 7.0000e-04
Epoch 14/50
1266/1266 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step - accuracy: 0.1522 - loss: 4.8874
Epoch 14: ReduceLROnPlateau reducing learning rate to 0.0004900000232737511.
1266/1266 ━━━━━━━━━━━━━━━━━━━━ 88s 69ms/step - accuracy: 0.1522 - loss: 4.8873 - val_accuracy: 0.1445 - val_loss: 6.5036 - learning_rate: 7.0000e-04
Epoch 14: early stopping
Restoring model weights from the end of the best epoch: 4.
Evaluating on test set...
158/158 ━━━━━━━━━━━━━━━━━━━━ 3s 21ms/step - accuracy: 0.1251 - loss: 6.5185

=== FINAL RESULTS ===
Test Loss: 6.5378
Test Accuracy: 0.1298
Test Perplexity: 690.7332
```



```
=== GENERATED TEXT ===
Temperature 0.8 (More coherent):
To be or not to a lord, And the own more of the own of a man of my king

Temperature 1.0 (Balanced):
To be or not to your man of your lord, And the good and the lord, of my king is
```

Hidden Units	Layers	Dropout	Test Acc.	Perplexity	Example Genera
600	(2 BiLSTM + 1 LSTM)	0.2	0.1265	729.46	“To be or not to not the good Gentleman: I am
256	(2 BiLSTM + 1 LSTM)	0.2	0.1274	690.99	“To be or not to the lord, I be the own good G
600	(2 BiLSTM + 1 LSTM)	0.3	0.1201	762.88	“To be or not to the man to the own of yo
600	(2 BiLSTM)	0.3	0.1269	629.21	“To be or not to the own own good good
600	(1 BiLSTM + 1 LSTM)	0.3/0.25	0.1298	690.73	“To be or not to a lord, And the own mo

3 Question 3: PixelRNN (PixelCNN , Row LSTM , Diagonal BiLSTM)

3.1 Introduction

The third experiment explores autoregressive generative models applied to the CIFAR-10 dataset. We implement and compare PixelCNN, Row LSTM, and Diagonal BiLSTM architectures, each designed to model image pixels sequentially. These models capture dependencies between pixels using masked convolutions or recurrent connections along rows and diagonals. We evaluate training loss, validation loss, and bits-per-dimension as metrics of generative quality.

3.2 Methodology

The following models were implemented in PyTorch:

- PixelCNN: Uses masked convolutions (type A and B) to ensure autoregressive conditioning. Residual connections and batch normalization were applied.
- Row LSTM: Processes images row-by-row using convolutional input-tostate and state-to-state transitions.
- Diagonal BiLSTM: Employs skewing/unskewing to process diagonals with bidirectional LSTMs.

All models were trained on CIFAR-10 with batch size = 64, Adam optimizer, and 5 epochs. Training was accelerated on GPU (device: cuda). Performance was reported in terms of training/validation loss and bits-per-dimension.

Table 3. Results of PixelCNN, Row LSTM, and Diagonal BiLSTM on CIFAR-10 (5 epochs).

Model	Final Train Loss	Final Val Loss	Bits/dim
PixelCNN	5.4132	5.4158	7.81
Row LSTM	5.1240	5.1378	7.41
Diagonal BiLSTM	5.4038	5.4041	7.80

3.3 Results

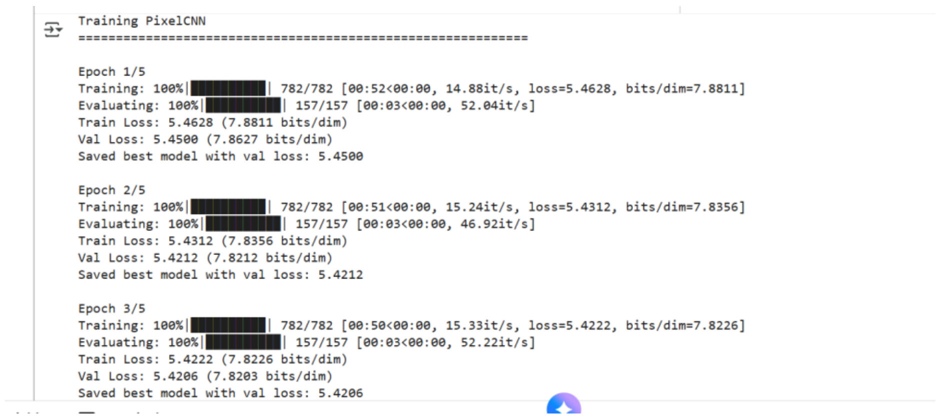
Table 3 summarizes the results for the three models.

3.4 Discussion

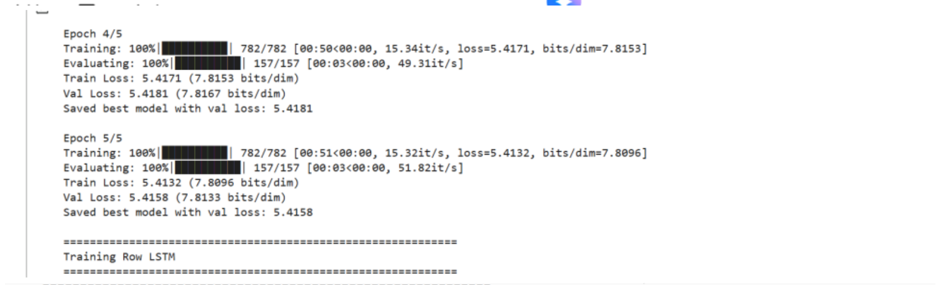
– Row LSTM achieved the lowest validation loss (5.1378) and best bits-perdim (7.41), confirming the advantage of row-wise recurrent modeling for capturing long-range pixel dependencies.
– PixelCNN provided competitive results but converged slower, with bitsper-dim around 7.81.
– Diagonal BiLSTM performed similarly to PixelCNN, but training time per epoch was significantly longer (7m37s vs. 52s for PixelCNN).
– Overall, Row LSTM demonstrated superior generative modeling performance on CIFAR-10, but with higher computational cost compared to PixelCNN.

4 Conclusion

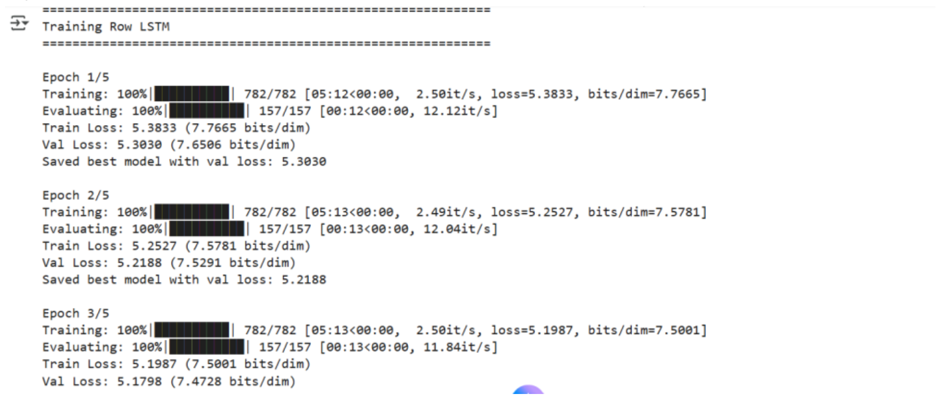
CNN performance on CIFAR-10 depended heavily on depth, filters, and learning rate, with the 7-layer CNN reaching the best accuracy (85.92%). For Shakespeare text generation, deeper BiLSTM-based models improved coherence but still showed high perplexity and repetition. In autoregressive image modeling, Row LSTM achieved the lowest bits per dimension, while PixelCNN offered faster training. Overall, results highlight the trade-offs between accuracy, coherence, efficiency, and modeling capacity across vision and language tasks.



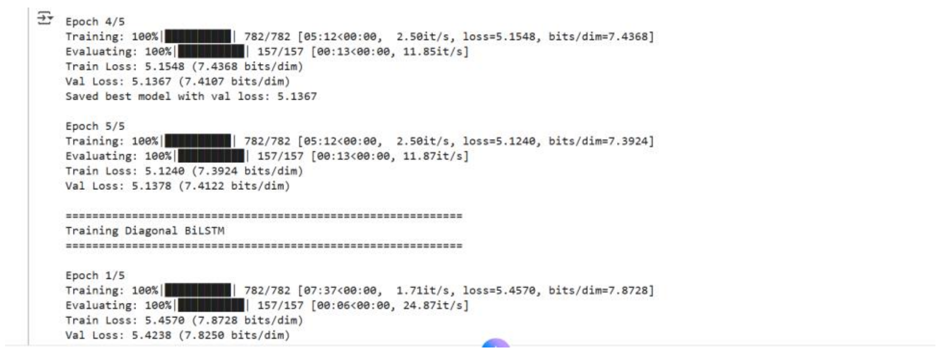
```
Training PixelCNN
============================================================

Epoch 1/5
Training: 100%|██████████| 782/782 [00:52<00:00, 14.88it/s, loss=5.4628, bits/dim=7.8811]
Evaluating: 100%|██████████| 157/157 [00:03<00:00, 52.04it/s]
Train Loss: 5.4628 (7.8811 bits/dim)
Val Loss: 5.4500 (7.8627 bits/dim)
Saved best model with val loss: 5.4500

Epoch 2/5
Training: 100%|██████████| 782/782 [00:51<00:00, 15.24it/s, loss=5.4312, bits/dim=7.8356]
Evaluating: 100%|██████████| 157/157 [00:03<00:00, 46.92it/s]
Train Loss: 5.4312 (7.8356 bits/dim)
Val Loss: 5.4212 (7.8212 bits/dim)
Saved best model with val loss: 5.4212

Epoch 3/5
Training: 100%|██████████| 782/782 [00:50<00:00, 15.33it/s, loss=5.4222, bits/dim=7.8226]
Evaluating: 100%|██████████| 157/157 [00:03<00:00, 52.22it/s]
Train Loss: 5.4222 (7.8226 bits/dim)
Val Loss: 5.4206 (7.8203 bits/dim)
Saved best model with val loss: 5.4206
```



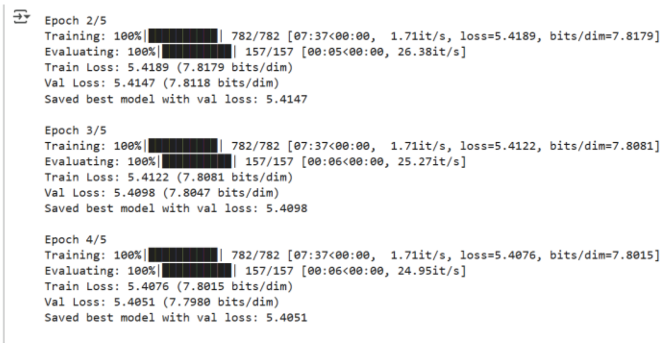
```
Epoch 4/5
Training: 100%|██████████| 782/782 [00:50<00:00, 15.34it/s, loss=5.4171, bits/dim=7.8153]
Evaluating: 100%|██████████| 157/157 [00:03<00:00, 49.31it/s]
Train Loss: 5.4171 (7.8153 bits/dim)
Val Loss: 5.4181 (7.8167 bits/dim)
Saved best model with val loss: 5.4181

Epoch 5/5
Training: 100%|██████████| 782/782 [00:51<00:00, 15.32it/s, loss=5.4132, bits/dim=7.8096]
Evaluating: 100%|██████████| 157/157 [00:03<00:00, 51.82it/s]
Train Loss: 5.4132 (7.8096 bits/dim)
Val Loss: 5.4158 (7.8133 bits/dim)
Saved best model with val loss: 5.4158

============================================================
Training Row LSTM
============================================================
```



```
============================================================
Training Row LSTM
============================================================

Epoch 1/5
Training: 100%|██████████| 782/782 [05:12<00:00,  2.50it/s, loss=5.3833, bits/dim=7.7665]
Evaluating: 100%|██████████| 157/157 [00:12<00:00, 12.12it/s]
Train Loss: 5.3833 (7.7665 bits/dim)
Val Loss: 5.3030 (7.6506 bits/dim)
Saved best model with val loss: 5.3030

Epoch 2/5
Training: 100%|██████████| 782/782 [05:13<00:00,  2.49it/s, loss=5.2527, bits/dim=7.5781]
Evaluating: 100%|██████████| 157/157 [00:13<00:00, 12.04it/s]
Train Loss: 5.2527 (7.5781 bits/dim)
Val Loss: 5.2188 (7.5291 bits/dim)
Saved best model with val loss: 5.2188

Epoch 3/5
Training: 100%|██████████| 782/782 [05:13<00:00,  2.50it/s, loss=5.1987, bits/dim=7.5001]
Evaluating: 100%|██████████| 157/157 [00:13<00:00, 11.84it/s]
Train Loss: 5.1987 (7.5001 bits/dim)
Val Loss: 5.1798 (7.4728 bits/dim)
```

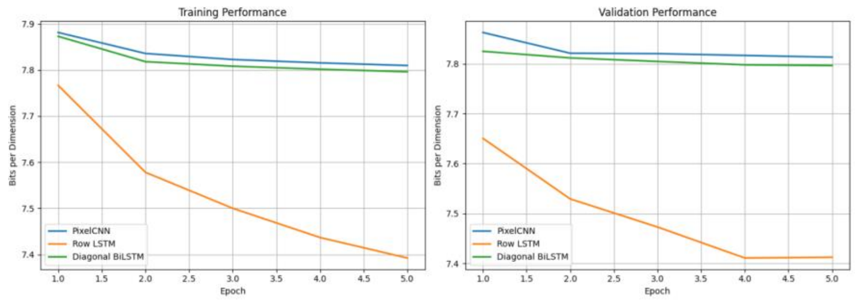


```
Epoch 4/5
Training: 100%|██████████| 782/782 [05:12<00:00,  2.50it/s, loss=5.1548, bits/dim=7.4368]
Evaluating: 100%|██████████| 157/157 [00:13<00:00, 11.85it/s]
Train Loss: 5.1548 (7.4368 bits/dim)
Val Loss: 5.1367 (7.4107 bits/dim)
Saved best model with val loss: 5.1367

Epoch 5/5
Training: 100%|██████████| 782/782 [05:12<00:00,  2.50it/s, loss=5.1240, bits/dim=7.3924]
Evaluating: 100%|██████████| 157/157 [00:13<00:00, 11.87it/s]
Train Loss: 5.1240 (7.3924 bits/dim)
Val Loss: 5.1378 (7.4122 bits/dim)

============================================================
Training Diagonal BiLSTM
============================================================

Epoch 1/5
Training: 100%|██████████| 782/782 [07:37<00:00,  1.71it/s, loss=5.4570, bits/dim=7.8728]
Evaluating: 100%|██████████| 157/157 [00:06<00:00, 24.87it/s]
Train Loss: 5.4570 (7.8728 bits/dim)
Val Loss: 5.4238 (7.8250 bits/dim)
```



```
Epoch 2/5
Training: 100%|██████████| 782/782 [07:37<00:00,  1.71it/s, loss=5.4189, bits/dim=7.8179]
Evaluating: 100%|██████████| 157/157 [00:05<00:00, 26.38it/s]
Train Loss: 5.4189 (7.8179 bits/dim)
Val Loss: 5.4147 (7.8118 bits/dim)
Saved best model with val loss: 5.4147

Epoch 3/5
Training: 100%|██████████| 782/782 [07:37<00:00,  1.71it/s, loss=5.4122, bits/dim=7.8081]
Evaluating: 100%|██████████| 157/157 [00:06<00:00, 25.27it/s]
Train Loss: 5.4122 (7.8081 bits/dim)
Val Loss: 5.4098 (7.8047 bits/dim)
Saved best model with val loss: 5.4098

Epoch 4/5
Training: 100%|██████████| 782/782 [07:37<00:00,  1.71it/s, loss=5.4076, bits/dim=7.8015]
Evaluating: 100%|██████████| 157/157 [00:06<00:00, 24.95it/s]
Train Loss: 5.4076 (7.8015 bits/dim)
Val Loss: 5.4051 (7.7980 bits/dim)
Saved best model with val loss: 5.4051
```

```
Saved best model with val loss: 5.4041
```


Training Performance
Validation Performance
Bits per Dimension
Epoch
PixelCNN
Row LSTM
Diagonal BiLSTM