<u>TASK - 2</u>

- ➔ Training STACKED autoencoders for coloured images.
- ➔ Input - (1 x 828) representation.
- ➔ Output - softmax probabilities for 5 - class(Highway, Forest, Inside city, Mountain, Street).
- ➔ Train, test - 80 : 20
- ➔ <u>Architecture details</u>:
    - ◆ 3 stackings, 2 fully connected layers (includes softmax).
    - ◆ AE1 - (828, 512),(512, 256) -- (256, 512), (512, 784)
    - ◆ AE2 - (256, 128), (128, 100) -- (100, 128), (128, 256)
    - ◆ AE3 - (100, 80), (80, 64) -- (64, 80), (80, 100)
    - ◆ FC1 - (64, 100)
    - ◆ FC2 - (100, 5)
    - ◆ Non-linearity - ReLu (total 7 = 6 + 1)
    - ◆ Optimizer - Adam
    - ◆ Loss - MSE (for encoder - decoder), CrossEntropy(for classifier)
    - ◆ Learning rate - 0.001
    - ◆ Stop criteria =   abs(cur_loss - prev_loss) < **0.001**
- ➔ <u>Training procedure</u> :
    - ◆ Train 3 encoder-decoder frameworks independently.
    - ◆ Pick encoders from them, append the fully connected layers and train entirely.
    - ◆ So the reduced representation is passed to fully connected layers while training the classifier.
- ➔ <u>OBSERVATIONS</u> :
    - ◆ Train accuracy - 100, Test accuracy - 73
    - ◆ Comparing task 1 and 2, we can see that the stacked autoencoder is able to get slightly better test accuracy(73%) than a single autoencoder(70%). This could be due to the fact that the 3 encoder-decoder frameworks are trained independently in the stacked mechanism.
    - ◆ When the random weights are initialized instead of the pretrained weights , the accuracy is 66.2, whereas after using pretrained weights the accuracy is 73.

<u>TASK - 3</u>

- ➔ Training STACKED autoencoders for black&white images.
- ➔ Input - (1 x 784) image.
- ➔ Output - softmax probabilities for 5 - class(Ankleboot, Trouser, T_shirt, Pullover, Sneaker)
- ➔ Train, test - 80 : 20
- ➔ <u>Architecture details</u>:
    - ◆ 3 stackings, 2 fully connected layers (includes softmax).
    - ◆ AE1 - (784, 512),(512, 256) -- (256, 512), (512, 784)
    - ◆ AE2 - (256, 128), (128, 100) -- (100, 128), (128, 256)
    - ◆ AE3 - (100, 80), (80, 64) -- (64, 80), (80, 100)

    - FC1 - (64, 100)
    - FC2 - (100, 5)
    - Non-linearity - ReLu (total 7 = 6 + 1), Tanh (just before softmax).
    - Optimizer - Adam
    - Loss - MSE (for encoder - decoder), CrossEntropy(for classifier)
    - Learning rate - 0.001
    - Stop criteria = abs(cur_loss - prev_loss) < **0.1**
- Training procedure :
    - Train 3 encoder-decoder frameworks independently.
    - Pick encoders from them, append the fully connected layers and train entirely.
    - So the reduced representation is passed to fully connected layers while training the classifier.
- OBSERVATIONS :
    - Train accuracy - 0.97, Test accuracy - 0.96.
    - While tanh gave accuracy in the range 0.20.
    - Due to stacking training is done with less computational complexity.

When the random weights are used the accuracy(0.95) obtained, which is almost the same as that of using pretrained weights.

TASK - 1

The task is to do image classification using a MLFFNN with deep CNN features for the images extracted from :

a)**VGGNet** - VGGNet is a pretrained model on the Imagenet data and it classifies the images into 1000 categories. Tensorflow Keras API provides the pretrained vgg16 model, in which the final layer was removed and a softmax layer was added with 7 output nodes for the 7 classes.

- Input - (224 x 224 x 3) size image.
- Output - softmax probabilities for 7 -classes, class(013.Bobolink, 034.Gray_crowned_Rosy_Finch, 041.Scissor_tailed_Flycatcher, 067.Anna_Hummingbird, 114.Black_throated_Sparrow, 142.Black_Tern, 170.Mourning_Warbler).
- Train, validation and test split - 70 : 10 : 20
- Pass the image to the pretrained VGGNet model with the modified output layer. The output node corresponding to the class with maximum probability is the predicted class.
- Architecture details:
    - VGGNet architecture with the final layer changed to output 7 classes instead of 1000.
    - Optimizer - Stochastic gradient descent(learning_rate=0.01)
    - Loss - Categorical Cross Entropy
    - Stopping criteria - EarlyStopping by monitoring the validation loss. Stops training when there is no improvement in the best validation loss seen so far, for 5 consecutive epochs.

    - Total params: 134,289,223
    - Trainable params: 28,679
    - Non-trainable params: 134,260,544
- Training procedure :
    - Train the model in batch mode using batch size=10, with early stopping on the validation loss as the stoppage criterion. The pretrained weights of the VGGNet were fixed and were marked as not to be modified/trained. Only the weights of the final layer were trainable.
    - Test the trained model on the test set.
- Training accuracy - 87.6%, Validation accuracy - 90.6%, Test accuracy- 79.5%
- Experiments: The pretrained **VGGNet** Model was also appended with multiple dense layers after removing the final dense layer of 1000 nodes. Dense layers of size 100,50 and a final output layer of size 7 were added and this model was trained to evaluate the performance. This **did not** give a higher performance than the model illustrated above since VGGNet has a very large number of parameters and adding more complexity was observed to overfit to the training data, hence reducing the test accuracy.

b)**GoogleNet** - GoogleNet is a pretrained deep CNN model on Imagenet based on the inception architecture. Since GoogleNet is not directly available in Keras, we made use of the **InceptionV3** architecture, which added factorizations, batch normalization etc. on top of GoogleNet.

- Input - (299 x 299 x 3) size image.
- Preprocessing function - tf.keras.applications.inception_v3.preprocess_input- It scales the pixel values in the range (-1,1), which is expected to be the input to the pretrained model.
- Output - softmax probabilities for 7 -classes, class(013.Bobolink, 034.Gray_crowned_Rosy_Finch, 041.Scissor_tailed_Flycatcher, 067.Anna_Hummingbird, 114.Black_throated_Sparrow, 142.Black_Tern, 170.Mourning_Warbler).
- Train, validation and test split - 70 : 10 : 20
- Pass the image to the pre-trained GoogleNet model with the MLFFNN attached to it. The output node corresponding to the class with maximum probability is the predicted class.
- Architecture details:
    - InceptionV3 architecture with the final dense layer removed which had 1000 nodes.
    - Appended a MLFFNN to the pre final layer of the above model, with a dense layer of 100 nodes, followed by an output layer of 7 nodes.
    - Optimizer - Stochastic gradient descent(learning_rate=0.01)
    - Loss - Categorical Cross Entropy
    - Stopping criteria - EarlyStopping by monitoring the validation loss. Stops training when there is no improvement in the best validation loss seen so far, for 4 consecutive epochs.
    - Total params: 22,008,391

- Trainable params: 205,607
- Non-trainable params: 21,802,784

➔ Training procedure :

- Train the model in batch mode using batch size=10, with early stopping on the validation loss as the stoppage criterion. The pretrained weights of the GoogleNet were fixed and were marked as not to be modified/trained. Only the weights of the last two dense layers were trainable.
- Test the trained model on the test set.

➔ Training accuracy - 90.1%, Validation accuracy - 95%, Test accuracy- 85.5%

➔ Experiments: Increasing the number of dense layers in the MLFFNN part of the model was observed to not increase the test accuracy since it was overfitting to the training data.

Comparison of VGGNet and GoogleNet models: From the above two analysis, we can see that GoogleNet(Inceptionv3) architecture performs better than VGGNet in terms of validation and test accuracy. This is because the number of parameters in GoogleNet is an order of magnitude less than VGGNet and also the GoogleNet model is much deeper than VGGNet. These two factors allow GoogleNet to generalize better and achieve better test accuracy in comparison to VGGNet, as is illustrated by the results above.

TASK - 2

Aim- To do Image classification using a CNN [CL1 + PL1, CL2 + PL2], MLFFNN.

➔ Input Resized - (224 x 224 x 3) size image.

➔ Output - softmax probabilities for 7 -classes, class(013.Bobolink, 034.Gray_crowned_Rosy_Finch, 041.Scissor_tailed_Flycatcher, 067.Anna_Hummingbird, 114.Black_throated_Sparrow, 142.Black_Tern, 170.Mourning_Warbler).

➔ Train, validation and test split - 70 : 10 : 20.

➔ Architecture details:

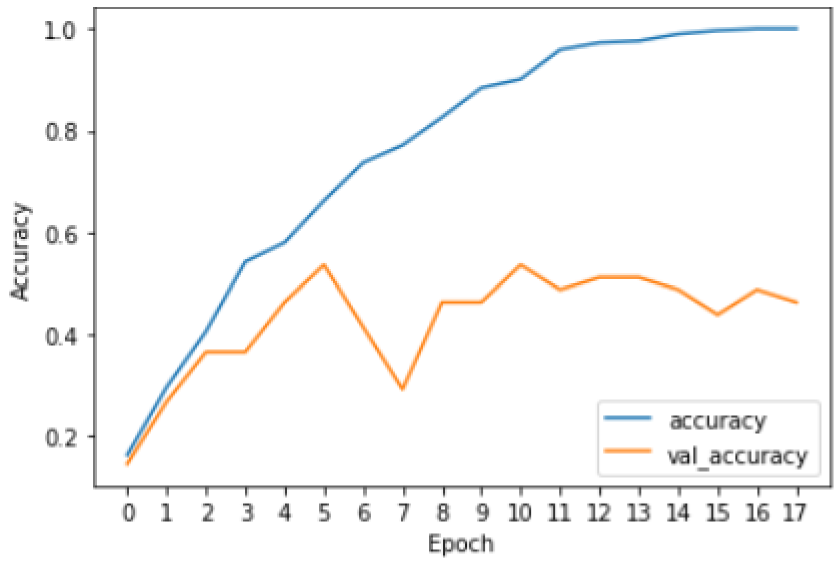
Layer	Number of Feature Maps	Kernel size	Kernel stride
Convolutional - 1	4	3x3	1
Convolutional - 2	16	3x3	1
Mean Pooling	_	2x2	2

Layer	Output shape	Traniable Parameters

Convolutional -1	222, 222, 4	112 (3*3*3+1)*4
Mean Pool - 1	111, 111, 4	0
Convolutional - 2	109, 109, 16	592 (4*3*3+1)*16
Mean Pool - 2	54, 54, 16 - 46656(flatten)	0
Fully connected - 1	128	5972096 (46656+1)*128
Softmax	7	903 (128+1)*7

- Optimizer - Adam(learning_rate=0.01)
- Activation function - Relu.
- Loss - Categorical Cross Entropy
- Stopping criteria - EarlyStopping by monitoring the validation loss. Stops training when there is no improvement in the best validation loss seen so far, for 5 consecutive epochs.
- Total parameters - 5,973,703 (all trainable).

RESULTS : Training accuracy - 100, Test accuracy - 48.23



INFERENCES : Due to less training data, the results are not good. Even after applying data augmentation, the results were not improved.

Task - 3

Machine translation, using Encoder - Decoder Framework.
Using LSTM, **without attention mechanism.**
Source_lang : English, Target_lang : Tamil
Glove embeddings are used for English.(`glove.6B.300d.txt`)

The tokenizer is trained on the glove words to create the vocabulary. For Tamil, the tokenizer is trained on the training corpus The training dataset consists of sentence pairs of (English,Tamil), split into batches of size 64. The sentences are converted into sequences where each number corresponds to the index of the word in the vocabulary. The sequence number corresponding to start and end token are also appended to each sentence. In every batch, the length of every sequence is equal to the longest sentence within that batch.

- **Encoder**
    - Input shape: (Batch_size = 64, padded_index_length), Embedding_matrix(eng_vocab_size, glove_embedding_size = 300) which remains constant throughout the training process.
    - LSTM(units)(Emb_output, Hidden_shape)
    - Output_shape : (Batch_size, padded_seq_length, enc_LSTM_units),
    - Hidden_shape : (Batch_size, enc_LSTM_units = 1024)
- **Decoder**
    - Input : Each sentence is passed word by word(Teacher_forcing).
    - Embedding : Keras_Embedding(tamil_vocab_size, emb_dim = 256)
    - LSTM(units)(Emb_output, (Enc_hidden,Enc_cell))
    - Output : next_dec_input, dec_hidden, dec_cell
    - Fully_connected_softmax_layer : (Batch_size, seq_length) X (targ_vocab_size)
- **Training Process**
    - Enc_input = Padded_Indices
    - enc_output, enc_hidden, enc_cell_____ = encoder(inp, enc_hidden)
    - Dec_input = (Batch_size, Start token = tamil_vocab_size+1)
    - So each time a word is predicted and loss is computed.
    - After passing the entire batch, the errors are back propagated to the encoder, decoder.
    - Trainable Parameters : enc_LSTMs, dec_LSTMs, dec_Embeddings, Fully_connected_softmax_layer.
    - Stop_criteria : loss_difference < 0.005 which took 40 - 50 epochs.
    - Loss_function : categorical cross entropy
    - Optimizer : Adam, Learning_rate = 0.001
    - Activation for LSTM : tanh, Recurrent Activation : sigmoid.
- Evaluation :
    - Each english_sentence is converted to word_index_sequence, start, end tokens are appended. Then it uses encoder, decoder trained weights.
    - The only difference in decoder while testing is it passes predicted_word as next dec_input, whereas while training, actual_word is used.
    - As the softmax layer will predict the most probable word, each time, the word_index_sequence of the target_vocab is generated, from which predicted_sentence in target_lang is generated.

The translation is evaluated using Bleu scores. The weights for 1-gram,2-gram,3-gram and 4-gram in Bleu at k=1,2,3,4 are defined as follows:

1.Bleu1: weights=(1, 0, 0, 0)
2.Bleu2: weights=(0.5,0.5,0,0)
3.Bleu3: weights=(0.33,0.33,0.33,0)
4.Bleu4: weights=(0.25,0.25,0.25,0.25)

The scores obtained after evaluating on the test set is as follows:

1.Bleu1: 0.092
2.Bleu2: 0.014
3.Bleu3: 0.004
4.Bleu4: 0.003

Task 4

Machine translation using encoder decoder framework.
Using LSTMs, **with attention mechanism.**
Source_lang : English, Target_lang : Tamil
Glove embeddings are used for English.(`glove.6B.300d.txt`)

- **Encoder**(same as previous)
    - Input shape: (Batch_size = 64, padded_index_length), Embedding_matrix(eng_vocab_size, glove_embedding_size = 300) which remains constant throughout the training process.
    - LSTM(units)(Emb_output, Hidden_shape)
    - Output_shape : (Batch_size, padded_seq_length, enc_LSTM_units),
    - Hidden_shape : (Batch_size, enc_LSTM_units = 1024)
- **Decoder**
    - Attention Part :
        - Scores = FC3(tanh(FC1(dec_hidden_with_time_axis = enc_hidden(init))+FC2(enc_output)))
          (FC3 input shape is (batch_size, pad_length, units),
          output : (batch_size, pad_length, 1))

        - Attention weights (Batch_size, pad_length, 1)= softmax(scores)
        - Context Vector (Batch_size, units) = attention_weights * Enc_output
    - Input : Each sentence is passed word by word(Teacher_forcing).
    - Embedding : Keras_Embedding(tamil_vocab_size, emb_dim = 256)
    - LSTM(units)(Emb_output, (Enc_hidden,Enc_cell))
    - Output : next_dec_input, dec_hidden, dec_cell
    - Fully_connected_softmax_layer : (Batch_size, seq_length) X (targ_vocab_size)
- **Training Process**

    - Enc_input = Padded_Indices
    - enc_output, enc_hidden, enc_cell = encoder(inp, enc_hidden)
    - Dec_input = (Batch_size, Start token = tamil_vocab_size+1)
    - Context_vector, Dec_hidden, Dec_cell are passed to dec_LSTM.
    - So each time a word is predicted and loss is computed.
    - After passing the entire batch, the errors are back propagated to the encoder, decoder(where attention weights are also updated).
    - Trainable Parameters : enc_LSTMs, dec_LSTMs, dec_Embeddings, attention weights, Fully_connected_softmax_layer.
    - Stop_criteria : loss_difference < 0.005 which took 40 - 50 epochs.
    - Loss_function : categorical cross entropy
    - Optimizer : Adam, Learning_rate = 0.001
    - Activation for LSTM : tanh, Recurrent Activation : sigmoid.
- Evaluation :
    - Each english_sentence is converted to word_index_sequence, start, end tokens are appended. Then it uses encoder, decoder trained weights.
    - The only difference in decoder while testing is it passes predicted_word as next dec_input, whereas while training, actual_word is used.
    - As the softmax layer will predict the most probable word, each time, the word_index_sequence of the target_vocab is generated, from which predicted_sentence in target_lang is generated.

  The translation is evaluated using Bleu scores. The weights for 1-gram,2-gram,3-gram and 4-gram in Bleu at k=1,2,3,4 are defined as follows:

  1.Bleu1: weights=(1, 0, 0, 0)
  2.Bleu2: weights=(0.5,0.5,0,0)
  3.Bleu3: weights=(0.33,0.33,0.33,0)
  4.Bleu4: weights=(0.25,0.25,0.25,0.25)

  The scores obtained after evaluating on the test set is as follows:

  1.Bleu1: 0.13
  2.Bleu2: 0.023
  3.Bleu3: 0.005
  4.Bleu4: 0.0031