1. Who are you (mini-bio) and what do you do professionally?

My name is Igor Ivanov. I'm deep learning engineer from Dnipro, Ukraine. I specialize in CV and NLP and work at a small local startup.

2. What motivated you to compete in this challenge?

First of all it was high quality dataset and very interesting task. Organizers did a great job collecting large and deeply representative set of satellite images. Also during this period I got support from TFRC program (tensorflow.org/tfrc) offering TPU resources to researchers. Given that I had enough time, compute, and the task directly related to my specialization and personal interest I just couldn't pass by.

3. High level summary of your approach: what did you do and why?

My final solution is an ensemble based on several different architectures (mostly CNN and CNN-LSTM) which were trained with different backbones and different depth of historical frames.

From the pure machine learning perspective the task is very interesting allowing application of several different approaches. The data is a set of time series of monochrome images depicting evolution of tropical storms. So from the beginning it was evident that there are 2 most important directions aiming to take advantage of:
1) SOTA convolutional models and their pretrained weights (e.g. EfficientNet) and
2) time-related information (i.e. to process not only single images but sequences of images).

Working on the 1st direction I trained model on 1-channel (monochrome) images as baseline. Then I trained model on 3-channel images. For this model I packed current frame and 2 previous frames as channels of a single RGB image. This approach gives much better result compared to monochrome images because it fully utilizes Imagenet pretrained weights and also model starts seeing some time-related information (because our channels are small part of time series). Developing this idea I tried to include historical images from different depths as channels. The best combination was step 9 i.e. [current_minus_18, current_minus_9, current]. When current frame is early in a sequence example is padded with 1st frame. For this class of models single input example is a 3-tensor of shape e.g. (366, 366, 3). The best model trained on this data was EfficientNet-B7.

As a variation of the 1st direction I also tried some models trained on RGB images where each channel is not a single monochrome frame but mean of some range of frames. Result were close but did not outperform approach with 3 individual frames.

Thinking about 2nd direction the first thing which comes to mind is to extract features from a series of images using CNN and then process extracted features with LSTM or GRU (https://arxiv.org/pdf/1411.4389.pdf). For the purpose of this approach I used already mentioned trick and packed monochrome frames as series of 3-channel RGB images. To process sequences of images I used TimeDistributed layer – a very efficient solution from tensorflow.keras. The best data configuration is 24 consequent monochrome frames in total (step 1) i.e. current frame and 23 previous frames. This data was represented as a sequence of 8 RGB images. Single input example is a 4-tensor of shape e.g. (8, 366, 366, 3). When current frame is early in a sequence example is padded with 1st frame. The best model architecture trained on this data was CNN with unidirectional LSTM, without attention, and ResNet50 backbone. EfficientNet-B3 as backbone gives very close result. I trained models on longer sequences up to 384. None of these

experiments outperformed my best model trained on 24 frames. In my experiments sequences longer than 96 gives much lower scores. Also I tried several different "time-processing heads" instead of LSTM all of which did not outperform regular LSTM:
1) Bidirectional LSTM with attention
2) GRU
3) Transformer consisting of 1, 6, 12 Encoder layers
4) ConvLSTM2D layer (https://arxiv.org/pdf/1506.04214.pdf)

Also to process sequential data I tried another interesting approach based on 3D convolution networks:
1) C3D (https://arxiv.org/pdf/1412.0767.pdf)
2) ResNet50-3D (https://github.com/ZFTurbo/classification_models_3D)
Results of these models were good enough but not better compared to previously mentioned approaches.

Backbones:

ResNet (50, 152)
SE-ResNet (50)
DenseNet (121)
Inception
InceptionResNet
Xception
EfficientNet (B0, B3, B7)
VisionTransformer (L32, L16, B16)

Best models and results:

My best submission (LB 6.2558) is ensemble of 51 models including representatives of all previously mentioned approaches trained with different backbones and different depth of historical frames. Smaller ensemble of 22 models (mostly subset of 51 models) can still reach 1st place with LB 6.3046 result. Some of my best single models by themselves can reach top10 in LB:
1) The best single model according to local cross-validation score (have no LB score), model id "run-20210126-0329"
    ResNet50 + LSTM trained on 24 frames
2) 7th place, LB 6.6216, model id "run-20210122-1753"
    EfficientNet-B0 + LSTM trained on 48 frames
3) 8th place, LB 6.6325, model id "run-20210121-1931"
    EfficientNet-B3 + LSTM trained on 24 frames

Other notes:

I used only images to train my models i.e. no other features (like time interval, or ocean of origin). I mostly used default resolution of 366x366, augmentations (flips and rotations) and corresponding TTAs. In all experiments I used inverted images (i.e. 255 - image). I trained everything with Adam optimizer. Batch sizes and learning rates were different with aim to have largest batch which can fit in memory. Learning rate schedule is simple reduction on plateau.

Mainly my models were solving regression task but also I trained several classification models (156 classes, softmax). Classification models were systematically worse compared to regression task, but they did contribute a bit to the final ensemble.

Ensemble optimization:

Ensemble is formulated as weighted arithmetic average optimized using pair-wise greedy search over sorted predictions.
Concept is the following:
1) Sort predictions from all models based on CV score in descending order. Smallest (i.e. best) value is last (predictions from each TTA are sorted independently)
3) Find best coefficients for pair of 1st and 2nd prediction, then apply these coefficients and compute result
4) Find best coefficients for pair of previous result and 3rd prediction, and so on

Experiment database

All experiments included in final ensembles share the same design and therefore scores are directly comparable. I collected scores and some parameters of my experiments in a single database. I hope it will make it easier to navigate over experiments and compute some statistics.

4. Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

1) First one is an implementation of my best CNN-LSTM architecture:

```
inp = Input(shape=(8, 366, 366, 3), dtype=tf.float32)
cnn = ResNet50(weights='imagenet', include_top=False)
gap2d = GlobalAveragePooling2D()
#
x = TimeDistributed(cnn)(inp)
x = TimeDistributed(gap2d)(x)
x = LSTM(1024, return_sequences=False)(x)
x = Dense(300, activation='relu')(x)
out = Dense(1, activation='linear')(x)
```

2) The 2nd important code fragment is data reading process for the mentioned best architecture. TFRecord files allow reading data substantially faster than raw images so I stored sequences of images as lists of byte strings in TFRecord files. During training reading and decoding process can be compactly implemented as following. I uses `tf.map_fn` to efficiently decode monochrome images.

```
feature_description = {
    'image': tf.io.FixedLenFeature([24], tf.string),
    'label': tf.io.FixedLenFeature([], tf.int64),}

def fn(image):
    image = tf.image.decode_jpeg(image, channels=1,
dct_method='INTEGER_ACCURATE')
    image = tf.image.resize(image, [366, 366])
    image = tf.cast(image, dtype=tf.uint8)
    return image

def parse_example(example_proto):
    d = tf.io.parse_single_example(example_proto, feature_description)
    #
    image = tf.map_fn(fn, d['image'], dtype=tf.uint8) # (24, 366, 366, 1)
    image = tf.squeeze(image)                         # (24, 366, 366)
    image = tf.reshape(image, [8, 3, 366, 366])       # (8, 3, 366, 366)
```

```
    image = tf.transpose(image, perm=[0, 2, 3, 1])    # (8, 366, 366, 3)
    label = tf.cast(d['label'], tf.int32)
    return image, label
```

3) And the 3rd one is to emphasize how it is easy to use mixed precision in tensorflow:

```
policy = tf.keras.mixed_precision.experimental.Policy('mixed_float16')
tf.keras.mixed_precision.experimental.set_policy(policy)
```

5. Please provide the machine specs and time you used to run your model.

All models were trained on the following configuration:
Ubuntu 18.04, 1 CPU, 6 GB RAM, TPUv3-8.
I tested some models on GPU equivalent:
Ubuntu 18.04, 16 CPU, 60 GB RAM, 8x V100 GPU, with mixed precision
This configuration can process default batch sizes I used on TPU and gives slightly higher training and inference speeds.
All experiments share the same setup: 5 folds, 10 predictions (1 original image and 9 test-time augmentations).
Training time depends on the architecture, and for CNN-LSTM it is highly dependent on sequence length.

CNN, EfficientNet-B7
Training time: 4 min per epoch of 56k examples
Training time per experiment: 8 hours (5 folds * 20 epochs * 4 min for training + inference)
Inference time for val set of 14k examples: 12 sec
Inference time for test set of 44k examples: 36 sec

CNN-LSTM, 24 frames, EfficientNet-B3 backbone
Training time per epoch: 20 min per epoch of 56k examples
Training time per experiment: 31 hours (5 folds * 15 epochs * 20 min for training + inference)
Inference time for val set of 14k examples: 1.6 min
Inference time for test set of 44k examples: 4.8 min

6. What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?

I trained several models from scratch i.e. without imagenet pretrained weights. All results were systematically worse compared to pretrained weights. Also I trained some models from scratch on 24-channel image (366, 366, 24). Result is substantially lower vs. 3 channels. Also I tried Bidirectional LSTM with attention. Result was close but did not contribute the final ensemble.

7. Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?

I did not use any special tool for data preparation or exploratory data analysis. As an EDA I just plotted sequences of frames for visual inspection.

8. How did you evaluate performance of the model other than the provided metric, if at all?

I used only RMSE as a metric. My local cross-validation setup is based on GroupKFold split by storm i.e. model always predicts completely unseen storms. This approach does not completely

mimic actual train/test split because there are some storms split by time between train and test. But cross-validation scores and leaderboard scores were very consistent.

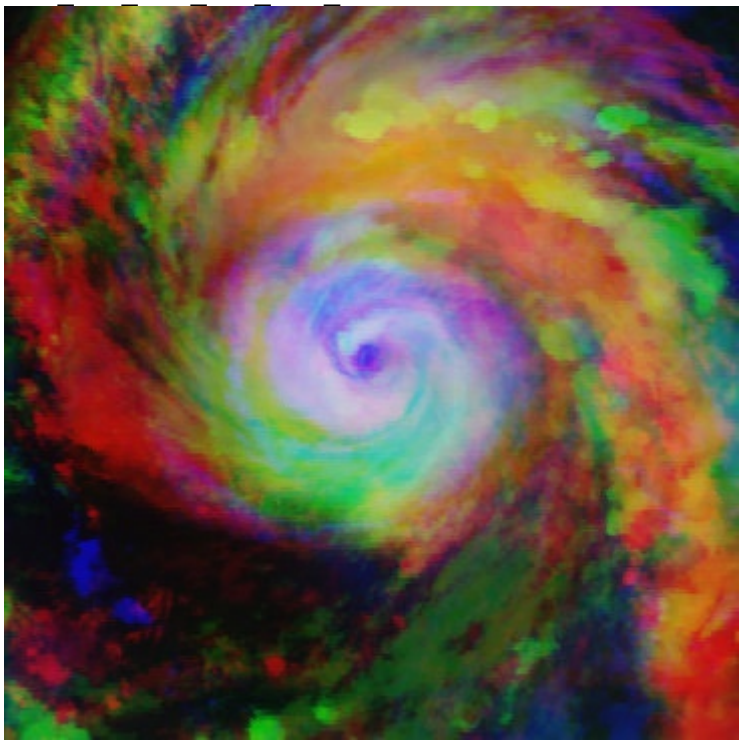
9. Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?

During my experiments models demonstrated stable results. If training or inference is performed on GPU mixed precision is good way to go. One important thing to keep in mind is that my training setup is based on 5 fold split (for validation and ensembling purposes) and 9 test-time augmentations. So effectively each model is trained 5 times and predicts 10 times (1 original image and 9 TTAs). In production settings training and inference time can be substantially reduced by training only one model (instead of 5) and predicting only original image (instead of TTAs).

10. Do you have any useful charts, graphs, or visualizations from the process?

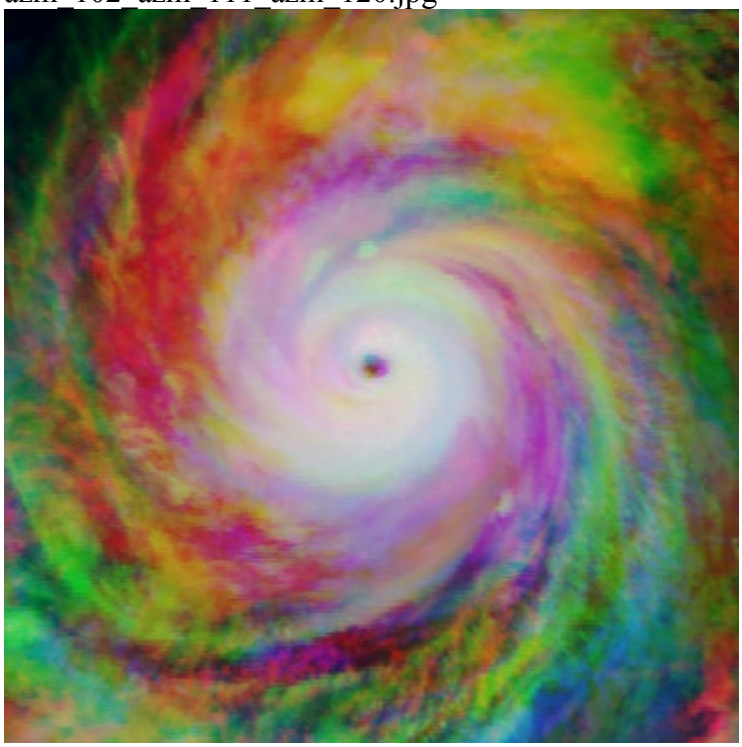
When we use color trick to pack monochrome frames into a single RGB image we get some beautiful images:

azm_062_azm_071_azm_080



azm_072_azm_081_azm_090.jpg

azm_102_azm_111_azm_120.jpg



11. If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?

I believe Transformers are very promising. I particular I did not spend much time tuning Transformer as a replacement of LSTM, but it worth to make some efforts in this direction.